Modified Selenium IDE and Test Management Application

By

Jared Hancock

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

University of Cincinnati
College of Education, Criminal Justice and Human Services

May 2012
Modified Selenium IDE and Test Management Application

By

Jared Hancock

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

© Copyright 2021 Jared Hancock

The author grants to the Information Technology Program permission
to reproduce and distribute copies of this document in whole or in part.

____________________________________________________________________  ____________
Jared Hancock                     Date

____________________________________________________________________  ____________
Hazem Said, PhD                  Date
Acknowledgements

I would like to thank Dr. Hazem Said for his patience and guidance on this project. In addition, I would like to thank Eric Schwab and the GE Aviation IDM team for providing the opportunity to use my co-op experience to define and begin to develop this project. Lastly, a special thanks to all of the contributors to the open source resources used in this project.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>i</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>1. Project Description and Intended Use</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem Statement</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Proposed Solution</td>
<td>3</td>
</tr>
<tr>
<td>2. Design Protocols</td>
<td>5</td>
</tr>
<tr>
<td>2.1 User Profiles</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Solution Details</td>
<td>7</td>
</tr>
<tr>
<td>2.3 User Interface</td>
<td>12</td>
</tr>
<tr>
<td>3. Deliverables</td>
<td>16</td>
</tr>
<tr>
<td>4. Project Planning</td>
<td>18</td>
</tr>
<tr>
<td>4.1 Schedule</td>
<td>18</td>
</tr>
<tr>
<td>4.2 Budget</td>
<td>18</td>
</tr>
<tr>
<td>5. Proof of Design</td>
<td>19</td>
</tr>
<tr>
<td>5.1 Selenium IDE Modification</td>
<td>20</td>
</tr>
<tr>
<td>5.2 Test Management Application</td>
<td>29</td>
</tr>
<tr>
<td>6. Testing</td>
<td>40</td>
</tr>
<tr>
<td>7. Conclusion and Recommendations</td>
<td>43</td>
</tr>
<tr>
<td>7.1 Conclusion</td>
<td>43</td>
</tr>
<tr>
<td>7.2 Recommendations</td>
<td>43</td>
</tr>
<tr>
<td>References</td>
<td>v</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Use Case Diagram</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2: Custom Export Options in Selenium IDE</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3: Partial Class Diagram</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4: Test Lifecycle Diagram</td>
<td>10</td>
</tr>
<tr>
<td>Figure 5: Dynamic Compiler Code Snippet</td>
<td>12</td>
</tr>
<tr>
<td>Figure 6: Run Tests Tab</td>
<td>13</td>
</tr>
<tr>
<td>Figure 7: Test Results Tab</td>
<td>13</td>
</tr>
<tr>
<td>Figure 8: Test Management Tab</td>
<td>14</td>
</tr>
<tr>
<td>Figure 9: Administration Tab</td>
<td>15</td>
</tr>
<tr>
<td>Figure 10: Credential Administration Page</td>
<td>16</td>
</tr>
<tr>
<td>Figure 11: Final Sprint Schedule</td>
<td>18</td>
</tr>
<tr>
<td>Figure 12: Project Budget</td>
<td>19</td>
</tr>
<tr>
<td>Figure 13: Custom Selenium IDE Export Options</td>
<td>20</td>
</tr>
<tr>
<td>Figure 14: gewebdriver.js Code Snippet</td>
<td>21</td>
</tr>
<tr>
<td>Figure 15: 1.js header/footer Formatter Snippet</td>
<td>22</td>
</tr>
<tr>
<td>Figure 16: 1.js header/footer Formatter Snippet</td>
<td>23</td>
</tr>
<tr>
<td>Figure 17 and 18: 1.js Code Snippet</td>
<td>24</td>
</tr>
<tr>
<td>Figure 19: Test Script Generated By IDMJUnitWebDriver Exporter</td>
<td>25</td>
</tr>
<tr>
<td>Figure 20: Test Script Generated By IDMJUnitWebDriver Exporter</td>
<td>26</td>
</tr>
<tr>
<td>Figure 21: Test Script Generated By IDMJUnitWebDriver Exporter</td>
<td>27</td>
</tr>
<tr>
<td>Figure 22: Test Script Generated By IDMJUnitWebDriver Exporter</td>
<td>28</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>23</td>
<td>Test Script Generated By IDMJUnitWebDriver Exporter</td>
</tr>
<tr>
<td>24</td>
<td>Initial Entry Into Test Management Application</td>
</tr>
<tr>
<td>25</td>
<td>Test Management Tab</td>
</tr>
<tr>
<td>26</td>
<td>Script Selected For Upload</td>
</tr>
<tr>
<td>27</td>
<td>Upload Successful</td>
</tr>
<tr>
<td>28</td>
<td>Script Available For Download</td>
</tr>
<tr>
<td>29</td>
<td>Download Script</td>
</tr>
<tr>
<td>30</td>
<td>Administration Tab</td>
</tr>
<tr>
<td>31</td>
<td>Selecting Script to Upload and Compile</td>
</tr>
<tr>
<td>32</td>
<td>Test Script Ready for Upload</td>
</tr>
<tr>
<td>33</td>
<td>Test Script Successfully Uploaded and Compile</td>
</tr>
<tr>
<td>34</td>
<td>Test Available to Run</td>
</tr>
<tr>
<td>35</td>
<td>Test Selected and Report Description Entered</td>
</tr>
<tr>
<td>36</td>
<td>Test Running In IE Browser</td>
</tr>
<tr>
<td>37</td>
<td>Test Report Available In Test Results Tab</td>
</tr>
<tr>
<td>38</td>
<td>Test Report View</td>
</tr>
<tr>
<td>39</td>
<td>Download Report</td>
</tr>
<tr>
<td>40</td>
<td>Downloaded Report</td>
</tr>
<tr>
<td>41</td>
<td>Table 1: Modified Selenium IDE Test Case</td>
</tr>
<tr>
<td>42</td>
<td>Table 2: Test Management Application Run Test Tab Test Case</td>
</tr>
<tr>
<td></td>
<td>Table 3: Test Management Application Test Results Tab Test Case</td>
</tr>
<tr>
<td></td>
<td>Table 4: Test Management Application Test Management Test Case</td>
</tr>
<tr>
<td></td>
<td>Table 5: Test Management Application Administration Test Case</td>
</tr>
</tbody>
</table>
Abstract

Application testing is a crucial part of the application development and maintenance process. Development teams must develop a testing plan and procedure that provides comprehensive coverage of their application while balancing the costs of manual testing, testing software licensing, automated test development, developer training, and testing software and hardware requirements. The Modified Selenium IDE and Test Management Application project was designed and developed to address these concerns on behalf of the GE IDM team. The project makes use of only free and open-source software, eliminating licensing costs. By utilizing a simple design with familiar controls and straightforward task separation, the project minimizes the need for specialized training for users of the software. In short, the project provides application test automation, eliminating the cost of manual testing without adding the traditional costs of commercial test automation software and training.
Modified Selenium IDE and Test Management Application

1. Project Description and Intended Use

1.1 Problem Statement

The GE Aviation and Energy Identity Management (IDM) team manages user data. The team manages, tracks, creates, updates, and deactivates accounts for integrated applications as well as managing approvals and auditing for Export Control (14). All of these responsibilities are accomplished by developing custom web applications which enable users to request, manage, and modify access depending on their individual user permissions.

The development of web applications creates the need for application testing to ensure required functionality, usability, interface design, compatibility, performance, and security (13). Throughout the application development industry, this testing is performed in multiple ways and to varying degrees of comprehensiveness. Some developers choose to do all testing manually. This form of testing best represents the way a user actually interacts with the application, but is very time consuming and is subject to human error. Many development teams, especially those working on applications of a broad scope, use some form of automated testing. Depending on their application development methodology, this may be accomplished by manually coding tests prior to application development, or as is the case with IDM, they may develop tests closer to the end of the application development cycle either by manually programming or through the use of a software test development application. Currently, there are many software testing tools available for use with web applications. These software packages range from free to paid, open source to proprietary, and function specific to comprehensive. All of these packages
offer different features and functionality, however the most widely used packages share the majority of the following features: Record and playback of tests, ability to parameterize the test values, script editor, test runner, and some form of run report (12). Among these tools is a Firefox plugin called Selenium IDE. During development of the third generation of IDM software applications (gen3) Selenium IDE was chosen as the preferred testing tool. This choice was made based on the facts that Selenium IDE is free, open source, has a user friendly test recording interface and is the most comprehensive of the free, open source applications considered by the team (7).

Selenium IDE had been a useful tool for IDM’s test development, but the team had noted some deficiencies. The primary deficiencies that were discovered include: Tests are only runnable in Firefox browser, browser navigation was not accurately represented, the runner did not record change of focus to dialog windows, and user credentials were stored as plain text (6)(7). Selenium IDE does offer export functionality with multiple export formats. However, further analysis of the exported test scripts exposed multiple unsupported test methods and commands which were not translated into code in the selected format.

The IDM team had begun the development of the fourth generation of IDM applications (gen4) and required a tool that can be used to create fully functional, comprehensive tests for approval, user acceptance, and regression testing. The tool had to enable non-developers to create test scripts in a click recording fashion. The tool needed to be free, open source, and available for commercial use. The tool was required to produce tests that will run on multiple browsers, Internet Explorer and Firefox at a minimum. The tests produced needed to be executable outside of the tool and be able to
be deployed to the application server for ease of maintenance and code versioning control. Tests had to support all functions currently available in end user applications as well as being editable to support future functions. Test reports needed to provide information that enables testers to quickly identify offending code.

Previously available testing tools that met the level of comprehensiveness required by the team did not meet other requirements. They were either paid applications, not available without fee for commercial use, or they required test developers to write testing code, or they did not produce tests that could be executed and modified outside of the tool and deployed to the application server. The Selenium IDE tool previously in use was the only free, open source tool that the team had been able to identify as approaching meeting the test plan requirements. Because it supported exporting test scripts in multiple programming languages, including the preferred Java language, this tool was chosen as the test development base tool for gen3 and gen4 applications (6)(7). However, the need still existed to meet test requirements not supported by the Selenium IDE tool.

In consideration of this need, I decided to develop a modified version of the Selenium IDE. The new application takes advantage of the Selenium IDE test recording interface, but export functionality was modified to produce fully functional tests which support all functions necessary to test all aspects of IDM applications. I developed a test management interface for deployment to the application server which includes the ability to view and download test result reports.

1.2 Proposed Solution
In response to the needs expressed by the IDM team for an application test development environment that would allow them to record, export, deploy, modify, and run automated web application tests, the proposed solution was a modified version of the Selenium IDE combined with a J2EE web application for performing test management functions.

The Selenium IDE source modifications were distributed to test developers with instructions for applying the modification to their current installation of Selenium IDE. Utilizing this modified plugin, the test developers can open a Firefox browser to the home page of the application under test and begin recording. The user facing portion of the recording interface remained unchanged. Following pre-written test cases, test developers click their way through the prescribed scenarios while the Selenium IDE captures every click and keystroke. After completing a test case, test developers are then able to click on the File menu and select “Export Test Case As”. At this point, the user is presented with a new option for export, IDMJUnit(WebDriver). Selecting this option will produce an un-compiled java file. This file is formatted as a JUnit test file utilizing the WebDriver API to control the browser interaction. The test developer then uploads this file to the testing server via the web interface.

The web user interface is the command center for the back end Java application that compiles and runs tests. The interface lists available test cases with the option to select any number of cases to run. Users also have the option to upload or download test scripts. Administrators have the added function of adding test scripts to the runnable test group. Users also have the ability to view and download result reports generated from test runs.
The Modified Selenium IDE and Test Management Software package has proven to be a proper response to the needs expressed by the IDM team on all facets. By extending the capabilities of the open source Selenium IDE project, this approach allowed the team to continue using the interface that they were already using and made it possible to convert the suite of tests that they had already developed to run with support for the previously unsupported functions. This eliminated any need for additional training on a new test recording system, while supporting the required test functionality. Tests recorded and exported using this solution are able to be executed in either Firefox or Internet Explorer, with the possibility to add support for additional browsers as needed. Reports generated from these test runs return precise information that can be used to pinpoint problem code in failing applications. As a replacement for manual testing, this application has reduced post deployment testing from what was often greater than 50 man-hours when no defects were found to an automated process that takes only minutes to start and requires no interaction to complete and report results. The time savings is even greater when defects are present as the savings compounds with each required test-fix-test cycle.(6)

2. Design Protocols

2.1 User Profiles

There are four types of users for this application. They are Test Developers, Testers, Administrators, and Application Approvers. All user types are IT professionals of various levels of experience. Some users hold more than one of the aforementioned roles.
Test Developers have previous experience in the use of Selenium IDE as a test recording application. Changes in this implementation are almost invisible to this user type. They have the added step of accessing the Test Management UI to upload tests, but this has proven not to be a task that is hard for software users of their experience level.

Administrators are the most technically savvy of the expected users. They have firm understanding of programming concepts and methods as well as experience with multiple programming languages and development environments. The Administrator’s role is to examine uploaded test scripts, make any necessary changes, including adding code necessary to make use of encrypted user credentials, and add approved tests to the runnable suite. The manual credential addition step is necessary due to security and export control compliance issues. (7)

The Testers group is comprised of application developers who use the application to test the functionality of web applications after changes are deployed. They have extensive experience with web applications from both a developer and end user standpoint. This user primarily selects a test or tests from the Web UI and runs the selection and then views the reports of the test run to determine if code changes “broke” any of the functionality of the application.

The least technical users are Application Approvers. Like the Testers, they select tests and run them and view the results. However, this user type is only concerned with whether the application passed or failed the test. It was necessary to keep this user in mind when designing the test reporting portion of the application as they do not
necessarily have the technical knowledge to interpret a test that uses a large amount of technical jargon.

2.2 Solution Details

There were four types of users that had to be considered when designing this application. Each user's roles are illustrated in the use case in Figure 1.

Use Case Diagram

![Use Case Diagram](image)

Figure 1

Much of the real work being performed by this application is behind the scenes as far as the end users are concerned. The complexity of the backend operations is masked by a straightforward, streamlined UI which breaks the test cycle down to record, upload,
run, view result. Test Developers continue to use the Selenium IDE to record tests as they perform them. These users have been using the Selenium IDE to record tests since the last implementation of gen3. The goal in the modification was to leave the existing Selenium UI as is and incorporate all changes in a way that did not change the user experience for this user type. This was accomplished by making modifications to the backend exporter scripts to support necessary functions without modifying the process of recording test scripts or performing an export. The only difference these users see with the Modified Selenium IDE is the additional export option in the export menu. (See Figure 2) In addition, this user interfaces with the Test Management Application to upload the exported test scripts.

Custom Export Options in Selenium IDE

![Custom Export Options in Selenium IDE](image_url)

Figure 2
In addition to the Modified Selenium IDE, we must discuss the design protocols of the Test Management Application. Since all expected users are experienced IT professionals with experience using multiple web applications the design goal was to create an experience that is intuitive enough that an experienced web application user can access the UI and perform the desired functions without any special training or instructions. This was accomplished by making use of the same interface look and feel as is implemented in the IDM Gen4 application. The application makes use of the PrimeFaces JSF library, which includes a wonderful set of styling skins which can easily be applied and exchanged to customize the look and feel without making programming changes or creating huge amounts of CSS. The main page of the UI is comprised of 4 tabbed sections. Each section is designed to accommodate a specific end goal as indicated by the tab title. The tabs include: Run Tests, Test Results, Test Management, and Administration. Each tab contains forms and/or controls which link to the Managed Beans of the backend Java application to perform the tasks prescribed by the given tab. In addition, the Administration includes controls for navigation to a separate page for credential administration. This separation makes it easier to use this option or remove it if it is not needed for a particular integration or if this task is managed by a separate application. The class structure of the Java application is designed to avoid ambiguity and follow an object oriented approach where each class represents an individual object or actor in the system. This design pattern is demonstrated by the design of the UI layer as illustrated in the class diagram shown in Figure 3.
The overall functionality and integration of Selenium IDE and the Test Management Application is best illustrated in terms of a Test life cycle. (See Figure 4)

Figure 3

Figure 4
Tests are recorded using the unchanged Selenium IDE recording interface. Completed tests are exported using the custom export option and then uploaded to the Test Management Application. Test Administrators then download the raw test, make any necessary changes and upload it to the runnable test section.

Of particular interest is the Dynamic Compiler node. To the user, this appears as a simple operation. I upload the file and it becomes available to run. However, this was a crucial piece of the design, the success or failure of which would determine the overall structure of the test lifecycle and subsequent design decisions. Figure 5 shows the code which ultimately performs the compilation. This code is contained in the uploadAndCompile method of the UploadAndCompile Managed Bean class. In order for this to properly execute, the jars being referenced had to be located in the deploy folder of the Jboss server. With the jars located in the proper folder, we are then able to add them to the classpath of the compilation task at runtime to make them available to the JavaCompiler in the current JVM. The successful completion of this compilation function eliminated the need for Administrators to compile tests along with the application and re-deploy in order to make added tests available. With this method, we can simply use reflection to load the test from an external file at runtime.
Upon initial entry into the application, the user is presented with the Run Tests tab by default. This tab was chosen as the default tab based on the fact that test execution will be the primary goal of users the majority of the time. The list of runnable tests is presented in a dual list design which allows the user to select and order tests by clicking on the test name and then clicking on the arrow, or by double-clicking the test name, or by drag and drop. This gives the user maximum control and caters to users with different operating schema in terms of previous experience. When a test name is clicked, the description of the test appears in the “Description of Selected Test” panel so the user can easily identify the correct test for their purposes. Once tests are selected and ordered the user can enter a name for the result file and click Run to execute the tests. If no name is entered for the Report, the user will be notified that this is a required field and asked to
try again. When the required fields are populated and the Run button is pressed the tests are executed and a result file is generated using the given title. The “Run Tests” tab is illustrated in Figure 6.

Run Tests Tab

![Run Tests Tab Image]

Figure 6

Test Results Tab

![Test Results Tab Image]

Figure 7
The second tab (Figure 7) is used for retrieving test results. It lists the available tests reports and the user is able to open the report by clicking on the report name. The Report view includes the option to download a copy of the report. The third tab (Figure 8) is used for uploading and downloading un-compiled test scripts. This section utilizes standard file upload and download controls that open a dialog box where the user will traverse their file system to select the file to upload or location to save the downloaded file. The upload and download controls are located in separate accordion style tabs. When the user clicks the header of the tab it toggles open and closed. The file upload section has an upload panel which gives the user the option to choose a file to upload. The user clicks on the Choose button and selects the file to upload. The user then types a description of the test in the description text field and selects the upload arrow. The file is then uploaded and the user is presented with a success message. The file download section presents a list of files available for download. The user selects the desired file and
clicks on the file name and is presented with the standard file save dialog to select the location to save to. The fourth and final tab (Figure 9) is for Administration options. This tab is present only for users who have administrative rights. In the current design, the primary purpose of the Administration tab is to allow the user to add a test to the list of runnables and add encrypted username/password pairs to the database. The user uploads the desired test in the same manner as on the Test Management tab. In the backend, the application dynamically compiles the .java file into a runnable .class file and appends the filename to the runnableTests.xml file used to populate the list on the Run Tests tab. This tab also includes a button to navigate to the credential administration page illustrated in Figure 10.

Administration Tab

![Administration Tab](image)

Figure 9
The majority of graphical style for this project is handled by the use of the PrimeFaces library. The PrimeFaces skin used was selected in consideration of the standard set of graphics and color schemes utilized by GE. This can easily be modified to give the application a customized appearance that fits into whatever environment in which it is to be employed.

3. Deliverables

Deliverables for this project consist of the following:

1) Modified Selenium IDE source code that presents the user with two new export options, IDMJUnit(WebDriver) and IDMJUnitInnerTest(WebDriver). The IDMJUnit(WebDriver) exporter produces a full script with supporting methods ready to be compiled and executed. The IDMJUnitInnerTest(WebDriver) exporter produces individual JUnit test scripts for insertion into full test scripts. The exporters provide support for the following methods in addition to methods already supported by the existing exporter:
a. Select
b. addSelection
c. isTextPresent
d. getTableCSS
e. getTableXPath
f. getTableByID
g. sendKeys

2) A Test Management Web Application written with HTML, Java/JSF utilizing the PrimeFaces library, and JDBC* which provides the ability to perform the following functions through the UI:
   a. Upload test scripts
   b. Download test scripts
   c. Add tests to runnable list
   d. Select tests from runnable list and run
   e. View and download test results
   f. Add/update username/encrypted password pair

3) A backend MYSQL* database which holds username/encrypted password pairs and related encryption keys.

*These two options have been modified in the final product. The developer experienced some difficulties with the database integration with the version of JBoss Server software being used (4.3.0). The project was ported over to Glassfish Server 3+ and integrated with a Derby database and JPA for improved portability.
4. Project Planning

4.1 Schedule

Figure 11 outlines the targeted schedule for this iteration of the project. This final sprint presented some technical difficulties, mostly due to the need to transfer the project to a new development machine and reconfigure the Jboss server. However, all issues were resolved and development was back on track by April 30th and completed as scheduled.

Final Sprint Schedule

![Final Sprint Schedule](image)

Figure 11

4.2 Budget
Due to the nature of this project, that is, the fact that the developer decided to make use of only open source or otherwise freely available software resources, the only budgetary concerns were for the needed hardware. Specifically, a Seagate 250 GB portable USB hard drive, which retails for approximately $50, but which was already owned and came at an actual incurred cost of $0. Additionally, development was being performed on a Dell laptop computer which was provided by GE Aviation. The retail cost of the computer was approximately $750. The computer was required to be turned in to GE at the end of the winter quarter. The developer purchased an equivalent Toshiba Satellite laptop computer to continue development. This came at a cost of $575. This budget is laid out in Figure 12. The developer secured a research grant in consideration of this project which was sufficient to cover the cost of the new laptop.

### Project Budget

<table>
<thead>
<tr>
<th>Item</th>
<th>Budget</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate 250 GB USB HDD-already owned</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td>Toshiba Satellite Laptop Computer</td>
<td>$750</td>
<td>$575</td>
</tr>
<tr>
<td>Jboss Server Software</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td>MySQL</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td>Eclipse IDE</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td>Selenium IDE</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$750</td>
<td>$575</td>
</tr>
</tbody>
</table>

Figure 12

### 5. Proof of Design
The product delivered at the end of this iteration meets all of the deliverables defined in the design freeze with some incidental modifications for improved operation and portability. This section gives an overview of how this was accomplished with the Selenium IDE modifications as well as the Test Management Application.

5.1 Selenium IDE Modifications

The Selenium IDE modifications are sufficient to support the required functions. In order to accomplish this, two of the source files utilized in the export of tests were modified to support export coverage for the needed functions. This provided both a custom export script (1.js) as well as a custom export controller (gewebdriver.js). In addition, a second export script was written which translates the test operations but does not write header and footer code in the exported script. This allows creation of a unit test which can be dropped into an existing full script. These two options are illustrated in Figure 13.

Custom Selenium IDE Export Options

![Custom Selenium IDE Export Options](image)

Figure 13

The export controller (See Figure 14) defines which methods will be supported by the exporter.
 gewebdriver.js Code Snippet

```javascript
//support for isTextPresent added by Jared
SeleniumWebDriverAdaptor.prototype.isTextPresent = function(elementLocator) {
    var locator = this._elementLocator('tagName:body');
    return WAPI.Util.isTextPresent(locator.type, locator.string, this.rawArgs);
};

//support for verifyTable added by Jared
SeleniumWebDriverAdaptor.prototype.verifyTable = function(elementLocator) {
    var locator = this._elementLocator(this.rawArgs[0]);
    if (locator.type == 'css'){
        var cssString = parseCSSString(locator.string);
        return WAPI.Util.getTableCSS(locator.type, cssString[0], cssString[1], cssString[2]);
    } else if (locator.type == 'xpath'){
        var xpathString = parseXPathString(locator.string);
        return WAPI.Util.getTable(xpathString[0], xpathString[1], xpathString[2]);
    } else if (locator.type == 'id'){
        var cssString = parseXPathString(locator.string);
        return WAPI.Util.getTableByID(locator.type, cssString[0], cssString[1], cssString[2]);
    }
    return WAPI.Util.getTable(locator.type, locator.string);
};

//support for select added by Jared
SeleniumWebDriverAdaptor.prototype.select = function(elementSelector) {
    var locator = this._elementLocator(this.rawArgs[0]);
    var index = this.rawArgs[1].indexOf('');
    return WAPI.Util.setSelected(locator.type, locator.string, this.rawArgs[1].slice(0, index), this.rawArgs[1].slice(index+1));
};

//support for addSelection added by Jared
SeleniumWebDriverAdaptor.prototype.addSelection = function(elementSelector) {
    var locator = this._elementLocator(this.rawArgs[0]);
    var index = this.rawArgs[1].indexOf('');
    return WAPI.Util.setSelected(locator.type, locator.string, this.rawArgs[1].slice(0, index), this.rawArgs[1].slice(index+1));
};
```

Figure 14

The export script provides both code translation for methods defined in the export controller as well as automated writing of class information and supporting methods as defined in the header and footer sections. (Figure 15-18)
1.js (maps to IDMJUnitWebdriver format) header/footer Formatter Snippet

```java
package $packageName;

import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.ui.Select;

public class $className {

private WebDriver driver;

private String baseUrl="";

private StringBuffer verificationErrors = new StringBuffer();

@Before

public void setUp() throws Exception {

tdriver = new InternetExplorerDriver();

tdriver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

}


public void $methodName() throws Exception {

}

@After

public void tearDown() throws Exception {

tdriver.quit();

}

public String verificationErrorsToString = verificationErrors.toString();

public void $methodName {

}

// verify some element exists on the page

private boolean isElementPresent(By by) {

}
```
Figure 16
Figure 17

1.js Code Snippet

```javascript
1019 UDAPI.Driver.searchContext = function(locatorType, locator) {
1020 var locatorString = xpathArgument(locator);
1021 switch (locatorType) {
1022 case 'xpath':
1023 return By.xpath(locatorString);
1024 case 'css':
1025 return By.cssSelector(locatorString);
1026 case 'id':
1027 return By.id(locatorString);
1028 case 'link':
1029 return By.linkText(locatorString);
1030 case 'name':
1031 return By.name(locatorString);
1032 case 'tagname':
1033 return By.tagName(locatorString);
1034 throw 'Error: unknown strategy [' + locatorType + ']' for locator [' + locator + ']';
1035 }
1036 // parses the search locators for tables
1037 UDAPI.Driver.tableSearchContext = function(locatorType, locatorStart, trStr, tdStr) {
1038 var locatorStringStart = xpathArgument(locatorStart);
1039 switch (locatorType) {
1040 case 'xpath':
1041 return By.xpath(locatorStringStart);
1042 case 'css':
1043 return By.cssSelector(locatorStringStart);
1044 case 'id':
1045 return By.id(locatorStringStart);
1046 throw 'Error: unknown strategy [' + locatorType + ']' for locator [' + locatorStart + ']';
1047 }
```

Figure 18
Figures 19-23 show a single test script generated utilizing this export controller and export script. It is important to note that this is a standard JUnit/Java script which utilized the WebDriver API. This means that it is not necessary to learn a specialized testing language in order to make any necessary modifications or additions to the script prior uploading it to the Test Management Application.

Test Script Generated By IDMJUnitWebDriver Exporter

```java
package webDriverRunner;

import java.util.regex.Pattern;

public class FirstTest {
    private WebDriver driver;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp() throws Exception {
        driver = new InternetExplorerDriver();
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

    @Test
    public void testFirst() throws Exception {
        driver.get("http://localhost:8080/AppTestInterface/");
        driver.findElement(By.linkText("Test Results")).click();
        driver.findElement(By.linkText("Test Management")).click();
        driver.findElement(By.linkText("Administration")).click();
    }

    @After
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }

    //verify some element exists on the page
    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        }
        catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

Figure 19
Test Script Generated By IDMJUnitWebDriver Exporter

```java
 } catch (NoSuchElementException e) {
    return false;
 }
//***begin custom methods***
//verify some string exists within the body tag
private boolean isTextPresent(By by, String toTest) {
    WebElement bodyTag = driver.findElement(by);
    if (bodyTag.getText().contains(toTest)) {
        return true;
    } else {
        return false;
    }
}
//verify table contains some string value
private String getTable(By by, String toMatch) {
    WebElement we;
    String text = "";
    try {
        we = driver.findElement(by);
        text = we.getText();
        if (text.contains(toMatch)){
            return toMatch;
        } else return text;
    } catch (NoSuchElementException e){
        return "element not found";
    }
}

//standard getTable method. returns the text contained in the table cell
private String getTable(By by) {
    try {
        WebElement we = driver.findElement(by);
        String text = we.getText();
        return text;
    } catch (NoSuchElementException e){
        return "element not found";
    }

Figure 20
```
private void setSelected(By by, String method, String value){
    Select select = new Select(driver.findElement(by));
    if (method == "label"){
        select.selectByVisibleText(value);
    } else {
        driver.switchTo().defaultContent();
    }
}

private String parseXPath(String xStart, String trString, String tdString){
    String nextNode = driver.findElement(By.xpath(xStart + "/*"));
    getTagName();
    if(!nextNode.equals("tr")){
        if(!((nextNode.equalsIgnoreCase("thead") & Integer.parseInt(trString) > 0)){
            int trCount = driver.findElements(By.xpath(xStart + "/*:not(\"tr\")")).size();
            String tdString = Integer.toString((Integer.parseInt(tdString) + 2));
            String xTD = "/[" + tdString + "]
            return xStart + xTR + xTD;
        } else {
            xStart = xStart + "+" + nextNode;
            String xStart = parseXPath(xStart, trString, tdString);
            return xStart;
        }
    } else {
        trString = Integer.toString((Integer.parseInt(trString) + 1));
        tdString = Integer.toString((Integer.parseInt(tdString) + 1));
        String xTR = "/[" + trString + "]
        String xTD = "/[" + tdString + "]
        return xStart + xTR + xTD;
    }
}

private String parseCSSPath(String cssStart, String trString, String tdString){
    String nextNode = driver.findElement(By.cssSelector(cssStart + " > "));
    getTagName();
    if(!nextNode.equals("tr")){
        //Returns the CSS path of the table cell
    }

Figure 21
Test Script Generated By IDMJUnitWebDriver Exporter

```java
111 if (!nextNode.equals("tr")) {
112     if (!nextNode.equalsIgnoreCase("thead") & Integer.parseInt(trString) > 1) {
113         int trCount = driver.findElements(By.cssSelector(cssStart + "+nextNode=" + ")").size();
114         nextNode = driver.findElement(By.cssSelector(cssStart + "+nextNode=" + ")").getTagName();
115         trString = Integer.tostring((Integer.parseInt(trString) - trCount));
116         cssStart = cssStart + "+" + nextNode;
117         cssStart = parseCSSPath(cssStart, trString, tdString);
118         return cssStart;
119     }
120     String cTR = "+";
121     for (int i = Integer.parseInt(trString); i > 0; i--) {
122         cTR = cTR + "+";
123     }
124     String cTD = "+";
125     for (int i = Integer.parseInt(tdString); i > 0; i--) {
126         cTD = cTD + "+";
127     }
128     return cssStart + cTR + cTD;
129 
130     // Returns the text contained in the table cell
131     private String getTableByID(String tableId, String trString, String tdString) {
132         try {
133             WebElement we = driver.findElement(By.id(tableId));
134             int trInt = Integer.parseInt(trString);
135             while (!we.findElement(By.xpath("/" + we.findElement(By.xpath("/" + we)).equals("tr")) {
136                 WebElement weTemp = we.findElement(By.xpath("/" + we));
137                 int trCount = weTemp.findElements(By.xpath("./" + we).size();
138                 we = we.findElement(By.xpath("./" + we);)
139                 trInt = trInt - trCount;
140                 if (weTemp.getTagName().equalsIgnoreCase("thead") & trInt > 0) {
141                     int trCount = weTemp.findElements(By.xpath("./" + we)).size();
142                     we = we.findElement(By.xpath("./" + we);
143                     trInt = trInt - trCount;
144                 } else {
145                     we = we.findElement(By.xpath("./" + we));
146                 }
147             }
148         } catch (NoSuchElementException e) {
149             return "element not found";
150         }
151     }
```

Figure 22

Test Script Generated By IDMJUnitWebDriver Exporter

```java
152     String xxPath = "/" + Integer.tostring(trInt + 1) + "/" + Integer.tostring(Integer.parseInt(tdString) + 1) + "/";
153     we = we.findElement(By.xpath(xxPath));
154     String text = we.getText();
155     return text;
156 } catch (NoSuchElementException e) {
157     return "element not found";
158 }
```
All of the complexity of these scripts is hidden from the end user who simply opens the Selenium IDE and presses record, interacts with the user interface, and then exports the recorded script. This means that a user is able to record a test utilizing the previously identified functions and export a test script that translates all of the functions to JUnit test language utilizing the WebDriver API.

5.2 Test Management Application

All required features are implemented in the Test Management Application. The application appearance is as discussed in section 2.3. The features implemented in this iteration of the Test Management Application include:

- User interface with four tabs: Run Test, Test Results, Test Management, Administration; and a second page for Credential Administration.
- Server side logic to handle upload/download of test scripts
- Server side logic to dynamically Compile tests into runnables list
- Server side logic to Run uploaded tests in desired browser and produce reports
- Application logic to view and download test result reports
- Derby database holding encrypted username/password pairs
- Application logic to enable upload and encryption of username/password pairs

This implementation provides the necessary functionality to meet all deliverable requirements. The Test Management Application provides full-featured management of
JUnit test scripts produced by the Selenium IDE export including the ability to upload and download test scripts, dynamically compile and execute tests as well as view and download test results. To demonstrate this, we will step through a full test lifecycle from the initial entry into the Test Management Application. We start with a test script recorded and exported using the IDMJUnitWebDriver formatter, Expo.java. Upon initial entry into the Test Management Application, we see that there are no tests available to run. (See Figure 24)

Initial Entry Into Test Management Application

![Initial Entry Into Test Management Application](image)

We select the Test Management tab (Figure 25) choose a test script to upload and include a description of the script. (Figure 26)
Test Management Tab

Figure 25

Script Selected For Upload

Figure 26
We press the upload arrow. The application uploads the file and informs us of the success. (Figure 27)

Upload Successful

![Upload Successful](image)

Figure 27

After a refresh, the script is now available for download for editing. (Figure 28)
We download the script (Figure 29) and make any necessary edits, such as ensuring there are no credentials being saved in plain text. In this case, there are none.
As a Test Administrator, we have the Administration tab available which will allow us to upload the script in a similar fashion as was done on the Test Management tab with the exception that the script is automatically compiled into the list of runnable tests. (See Figure 30-33)

Administration Tab
Selecting Script to Upload and Compile

**Figure 31**

Test Script Ready for Upload

**Figure 32**
Test Script Successfully Uploaded and Compiled

Figure 33

After a refresh, the test is now available to run on the Run Tests tab. (Figure 34) We click on the test and the description appears in the Description of Selected Test panel. We verify that this is the test we wish to run and move it to the To Run panel, enter a description for the Report and select Run.

Test Available to Run

Figure 34
Because we are testing on the same machine that is serving the application, we have the advantage of seeing the test execute. In a production environment, this would not be the case, but it serves us well for demonstration purposes. In this case, the test is testing this application in an IE browser. (Figure 36)
The test completes and after a refresh, the test is now available in the Test Results tab. (Figure 37) We click on the report name, the report is displayed as well as is the option to download the report. (Figure 38)

Test Report Available In Test Results Tab

![Test Report Available In Test Results Tab](image)

Figure 37

Test Report View

![Test Report View](image)

Figure 38
If desired, we can download a more detailed copy of the report as shown in figure 39. The downloaded report is dynamic, so we can show or hide elements and print the report with the desired information. (Figure 40)

Download Report
6 Testing

Testing was performed according to specific test scenarios for each tab of the Test Management Application as well as for the Modified Selenium IDE. The test scenarios included practical functional testing of all implemented features as modeled in Tables 1-5. These scenarios reflect some changes made to the UI design in consideration of providing a more intuitive, streamlined design. The tests and test scripts shown in figure 6 illustrate actual successful generation, upload and compilation of test scripts using the Modified Selenium IDE and Test Management Application. The test reports shown in figures 7, 39 and 40 demonstrate actual reports generated by the application.

Modified Selenium IDE Test Case

<table>
<thead>
<tr>
<th>Step #</th>
<th>Action</th>
<th>Data / Input</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Precondition</td>
<td>1. Log in</td>
<td>URL: <a href="http://IDMURL">http://IDMURL</a></td>
<td>The user successfully logged into IDM application using Firefox browser and opened Selenium IDE.</td>
</tr>
<tr>
<td>Scenario 1 - Verify exporter support for required functions.</td>
<td>2. Login with valid username and Password.</td>
<td>NA</td>
<td>Selenium IDE will begin recording user interactions with browser.</td>
</tr>
<tr>
<td>1. Precondition</td>
<td>2. Begin recording script by clicking on the record button in Selenium IDE.</td>
<td>NA</td>
<td>Select command will be added to selenium script</td>
</tr>
<tr>
<td>2. Precondition</td>
<td>3. Select Item in (select box control)</td>
<td>NA</td>
<td>Select command will be added to selenium script</td>
</tr>
<tr>
<td>4.1 Precondition</td>
<td>4. Right click on some text and select &quot;Verify Text ...&quot;</td>
<td>NA</td>
<td>verifyText command will be added to selenium script</td>
</tr>
<tr>
<td>4.2 Precondition</td>
<td>4. Right click on a table element and select Verify table with id locator</td>
<td>NA</td>
<td>verifyTable command will be added to selenium script with id locator</td>
</tr>
<tr>
<td>4.3 Precondition</td>
<td>4. Right click on a table element and select Verify table with XPath locator</td>
<td>NA</td>
<td>verifyTable command will be added to selenium script with XPath locator</td>
</tr>
<tr>
<td>4.3 Precondition</td>
<td>4. Right click on a table element and select Verify table with CSS locator</td>
<td>NA</td>
<td>verifyTable command will be added to selenium script with CSS locator</td>
</tr>
<tr>
<td>5.1 Precondition</td>
<td>5. Type keys into an autocomplete textbox</td>
<td>NA</td>
<td>Type command will be added to selenium</td>
</tr>
<tr>
<td>5.2 Precondition</td>
<td>5. In Selenium IDE click on dropdown beside &quot;type&quot; command and select typeKeys</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>6.1 Precondition</td>
<td>6.1 In Selenium IDE Select &quot;File--&gt;Export test case as&quot;&gt;&quot;IDMJUnitWebDriver&quot;</td>
<td>NA</td>
<td>The save as dialog box will be opened.</td>
</tr>
<tr>
<td>6.2 Precondition</td>
<td>6.2 type a name into the &quot;File name&quot; and select save &quot;SeleniumTest&quot;</td>
<td>NA</td>
<td>The test script will be exported to the selected location.</td>
</tr>
<tr>
<td>7. Precondition</td>
<td>7. Open the exported file and verify commands were translated and no &quot;Unsupported Command&quot; messages are present.</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

Table 1
### Test Management Application Run Test Tab Test Case

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>URL</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Log in</td>
<td></td>
<td>The user will be redirected to the User Interface home page. Run Tests tab will be displayed to the user as a default initial active tab.</td>
</tr>
</tbody>
</table>

#### Scenario 1 - Verify Test Description Panel

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Data / Input</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on any test in the &quot;Available&quot; panel.</td>
<td>NA</td>
<td>The Test Description will appear in the description panel. Pass</td>
</tr>
<tr>
<td>2</td>
<td>Mouse over any test name in the &quot;Available&quot; panel.</td>
<td>NA</td>
<td>The Test Description will appear as a tooltip. Pass</td>
</tr>
<tr>
<td>3.1</td>
<td>Move a test from the &quot;Available&quot; panel to the &quot;To Run&quot; panel</td>
<td>NA</td>
<td>The Test Description will appear as a tooltip. Pass</td>
</tr>
<tr>
<td>3.2</td>
<td>Click on a test in the &quot;Available&quot; panel.</td>
<td>NA</td>
<td>The Test Description will appear in the description panel. Pass</td>
</tr>
<tr>
<td>3.3</td>
<td>Click on the test in the &quot;To Run&quot; panel.</td>
<td>NA</td>
<td>The Test Description will appear in the description panel. Pass</td>
</tr>
</tbody>
</table>

#### Scenario 2 - Verify Run

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Data / Input</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Move a test to the &quot;To Run&quot; panel</td>
<td>NA</td>
<td>Pass</td>
</tr>
<tr>
<td>1.2</td>
<td>Enter a name for the test result file in the Report Title field.</td>
<td>NA</td>
<td>Pass</td>
</tr>
<tr>
<td>1.3</td>
<td>Click the &quot;Run&quot; button.</td>
<td>&quot;exampleReport&quot;</td>
<td>The test will be executed and the report will become available in the Results list on the Test Results tab. Pass</td>
</tr>
</tbody>
</table>

### Test Management Application Test Results Tab Test Case

<table>
<thead>
<tr>
<th>Step #</th>
<th>Action</th>
<th>Data / Input</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>1. Log in 2. Login with valid username and Password.</td>
<td>URL: <a href="http://localhost:8080/AppTestInterface/">http://localhost:8080/AppTestInterface/</a></td>
<td>The user will be redirected to the User Interface home page. Run Tests tab will be displayed to the user as a default initial active tab.</td>
</tr>
</tbody>
</table>

#### Scenario 1 - Verify test results retrieval

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Data / Input</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click on the &quot;Test Results&quot; tab.</td>
<td></td>
<td>The test results tab will be opened. The list of available reports will be present in the Test Reports listbox. Pass</td>
</tr>
<tr>
<td>1.1</td>
<td>Click on a test result name in the &quot;Test Reports&quot; panel.</td>
<td>NA</td>
<td>The report will be displayed along with a &quot;click to download&quot; link. Pass</td>
</tr>
<tr>
<td>1.2</td>
<td>Click the &quot;click to download&quot; link</td>
<td>NA</td>
<td>User will be presented with the download dialog to select a location. Pass</td>
</tr>
<tr>
<td>1.3</td>
<td>Save the file to your preferred location.</td>
<td>NA</td>
<td>The Report will be save to the users selected location. Pass</td>
</tr>
</tbody>
</table>

---

Table 2

Table 3
During the testing phase of the second iteration, this developer discovered additional functions which are not captured by the Selenium IDE. These fall outside the
scope of this project as they are not functions of controls present in the IDM Gen4 application. It seems necessary to mention, however, that these unsupported functions exist and may require additional research and development in order to apply this product to testing of other applications.

7 Conclusions and Recommendations

7.1 Conclusion

At the beginning of this project release night testing was a grueling manual process for the IDM development team. Repeated test-fix-test cycles often required the entire team of 8-10 developers to spend up to 8 hours testing over a release for a total of 80 man-hours of testing. This is only a snapshot of the testing that is performed over the development life-cycle. On a single release night, execution of automated tests produced using the Modified Selenium IDE performed side-by-side with manual testing covered the same testing scenarios. The manual testing took 50 man-hours. The total amount of time required to run the automated tests was less than 2 hours. With the completion of the Test Management Application, the automated testing time investment is reduced to minutes. The tests run unattended and provide a full report of results. The Modified Selenium IDE and Test Management Application is a powerful solution for the IDM team as well as for any team needing to streamline their testing process.

7.2 Recommendations

The final iteration of this project took place after the developer’s co-op employment with GE came to an end. This led to the exploration of how this project could benefit not only the GE IDM team, but the wider development community. With
this larger goal, the developer would recommend further expansion of the administrative options in the Test Management Application. One example would be to make it possible to upload jar files from the Administration tab that would then be able to be referenced by the dynamic compiler. This would expand the ability to increase test coverage without requiring redeployment of the application.

Working with open source software has proven to be both challenging and rewarding. In some cases, the lack of comments and directions caused painstakingly frustrating deciphering sessions, but in the end forced the developer to increase his knowledge of the technology and come to a better understanding of how the software worked. This developer would encourage any hopeful software developer to take on at least one challenge involving integration of open source software. It will surely prove to be both educational and bring about a great sense of accomplishment when the project is complete.
References

11. JUnit.org Resources for Test Driven Development. <www.junit.org>