

# Kuali Test Framework

Rob Tucker – University of Arizona

(revised)

07/12/2014

## Table of Contents

Overview.....	2
Configuration and Storage.....	3
Security.....	3
Application Architecture.....	3
Logical Test Model.....	3
Getting Started.....	5
Kuali Test Framework Configuration XML.....	5
Test Creator Command Line.....	6
Test Runner Command Line.....	6
Window Specific Proxy Setup Requirements.....	6
Test Creator User Interface.....	7
Test Creator Tool Bar and Main Menu.....	7
Initial Setup.....	9
Application Email.....	9
JDBC Database Connection.....	10
JMX Connection Setup.....	10
Web Service Connection Setup.....	11
Platform Setup.....	11
Additional Database Information.....	12
Creating a New Test.....	13
Creating a File Test.....	14
Completing Checkpoint Dialog.....	15
Creating a Web Service Test.....	16
Creating a Database Test.....	18
Selecting Columns for Database Query.....	20
Database associated with test platform.....	20
Database Table.....	21
Table Column.....	21
Primary Key Table Column.....	21
Building SQL Select and Order By Clause.....	22
Building SQL Where Clause.....	24
Global lookup by column name:.....	25
Column specific lookup.....	25
Displaying Generated SQL and Verifying Validity.....	26
Creating SQL Checkpoint.....	28
Creating a Web Application Test.....	29
Web Application Test Framework Overview.....	29
Test Creator.....	29
Test Runner.....	30
Starting a Web Application Test.....	31
Creating a Web Checkpoint.....	33
Default text input tag handler.....	35
Capital Asset Tab System Selection tag handler.....	36
Creating Web Checkpoint Properties.....	38

Creating Web Test Execution Parameters.....	39
Test Results Delivery.....	42
Global .....	42
Platform .....	42
Test Suite.....	43
Example Test Result.....	43
Test Management Panel.....	44
Repository.....	45
Platform Popup Menu.....	45
Test Suite Popup Menu.....	45
Test Popup Menu.....	45
Databases.....	46
Web Services.....	46
JMX Connections.....	46
Test Scheduling.....	47
Appendix A: Tag Handler Configuration.....	49
Tag Handler Java Classes.....	50
Interface.....	50
Abstract Base Class.....	50
General Purpose Handlers.....	50
Tag Handler XML Configuration.....	50
Tag handler configuration file naming convention.....	50
Tag Handler XML configuration example.....	51
Appendix B: Additional Database Information.....	55
Additional Database Information File Example.....	55
Appendix C: For Developers.....	57
Open Source Technologies Used.....	57
Libraries that Require Manual Addition to Maven Repository.....	57
Native Browser Libraries.....	57

## Overview

The Kuali Test Framework provides a simple interface for creating, saving and running application tests that closely replicate real-world application usage scenarios. Once created, a test runner provides functionality to run tests on a scheduled basis. The test runner supports multi-threaded, parallel test execution to replicate multi-user environments or drive load test scenarios. Test results are delivered via email with excel spreadsheet attachments.

The test framework is primarily targeted at web application testing. “Kuali Test Framework” is a bit of a misnomer as the application is designed as a configurable test environment that can support any web application. While primarily designed for web application testing, the framework supports several test methods to provide a full suite of test capabilities – supported test types include:

- HTTP/HTML based web application tests
- SQL query database tests
- File based tests

- SOAP based web service tests
- JMX memory tests

The primary goals of the Kuali Test Framework are as follows:

- Provide a simple interface that allow non-technical users to create tests during normal application user testing and evaluation.
- Support test scenarios that closely replicate real-world application usage.
- Support multi-user and load test scenarios.
- Provide an easily configured environment for test creation, scheduling and results delivery.
- Provide an application that can be configured to support complex web-based HTML interfaces while still presenting a standard, simple user interface for test creation.

## **General Description**

### ***Configuration and Storage***

XML documents are used throughout the application for configuration and storage. Excel files are used for test results.

### ***Security***

The application as it currently exists has no user authentication/authorization built in. Data security is handled via encryption using **Jasypt** (<http://www.jasypt.org>). Database, JMX and Web Service tests may requires passwords, if so, the passwords will be stored in an encrypted form in the XML configuration. Web tests may store sensitive information in XML test files that will later be passed as parameters during test execution. The application provides the capability to define parameter names requiring encryption and the values associated with these parameters will be encrypted prior to saving in the XML test documents.

### ***Application Architecture***

The Kuali Test Framework consist of 2 Java applications:

1. Swing based GUI front end for test environment management, test creation and test scheduling.
2. Java command line application designed to run as service for test execution and test results delivery.

### ***Logical Test Model***

The diagram below displays the logical model of the test application:

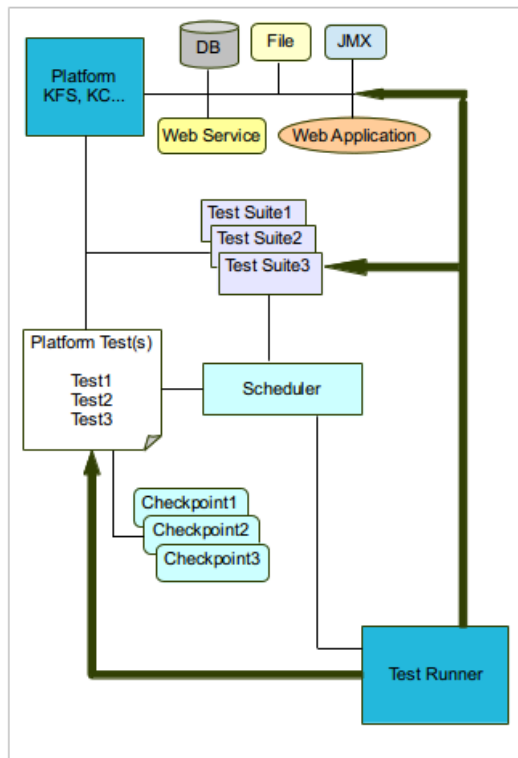


Diagram symbols are described below:

- **Platform** – defines the basic test foundation. A platform consists of an application of a specified version in a specified environment. For example DEV KFS 3.0, TST KC 5.0 etc. Tests, Test Suites, DB connections, JMX connections, Web Service URLs and Web Application URLs are associated with a specific platform.
- **Platform Test** – Tests are created for a specified platform, a test can be one of the following types – web, database, file or web service.
- **Checkpoint** – A checkpoint is composed of 1 or more checkpoint properties that are used to evaluate expected vs. actual test values for comparison purposes. During test creation check points are created and saved with expected (good) values. When tests are run via the scheduler the runtime values are evaluated against expected values to determine test status. Database, File and Web Service tests only support check points of the same type. A Web Application test supports check points of all types.
- **Test Suite** – Platform tests can be grouped together as a Test Suite which will execute in a single test execution context.
- **DB** – A platform can have an associated JDBC database connection which can then be used to create SQL tests and check points.
- **File** – The framework supports file tests and check points.
- **JMX** – A platform can have an associated JMX (Java management extension) connection to support server-side jvm memory check points.
- **Web Service** – A platform can have an associated web service connection (URL to WSDL) to support web service tests and checkpoints.

- **Web Application**– A platform can have an associated web application URL to support web application test creation.

## Getting Started

Below you will find the preliminary configuration required to use the Kuali Test Framework.

### *Kuali Test Framework Configuration XML*

Kuali Test Framework configuration is defined in an XML document whose path is passed to the application as an input parameter. An example test configuration file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<kuali-test-configuration xmlns="test.kuali.org">
  <repository-location>
    my-test-repository-root-directory
  </repository-location>
  <test-result-location>
    my-test-results-root-dir
  </test-result-location>
  <tag-handlers-location>
    my-html-tag-handler-definition-file-dir
  </tag-handlers-location>
  <default-test-wait-interval>2</default-test-wait-interval>
  <additional-db-info-location>
    my-additional-db-definition-file-dir
  </additional-db-info-location>
  <database-connections>
    <!-- database connections created through the user
      interface will go here -->
  </database-connections>
  <web-services>
    <!-- web service URLs setup through the user interface
      will go here -->
  </web-services>
  <jmx-connections>
    <!-- JMX URLs setup through the user interface
      will go here -->
  </jmx-connections>
  <modified>>false</modified>
  <email-setup>
    <mail-host>mail-host</mail-host>
    <subject>default-mail-subject</subject>
    <from-address>default-from-address</from-address>
    <to-addresses>default-to-address</to-addresses>
  </email-setup>
  <platforms>
    <!-- platforms defined through the user interface
      will go here -->
  </platforms>
  <test-execution-parameter-names>
    <!-- test execution parameter names defined through
      the user interface will go here -->
  </test-execution-parameter-names>
  <parameters-requiring-encryption>
```

```

        <!-- test parameter names requiring encryption
            defined through the user interface will go here -->
    </parameters-requiring-encryption>
    <encryption-password-file>
        path-to-file-containing-password-to-facilitate-encryption
    </encryption-password-file>
    <auto-replace-parameters>
        <!-- auto replace parameter names defined through
            the user interface will go here -->
    </auto-replace-parameters>
</kuali-test-configuration>

```

The XML above is the base configuration required to run the Kuali Test Framework. Please replace any file path entries with the the desired path. Tags that include comments with “defined through the user interface” will be populated using the test creator GUI.

### ***Test Creator Command Line***

Below is an example of the command line to start and run the test creator GUI application. It is expected that the kualitest-1.0.jar can be found on the classpath.

```

java Dhttp.proxyHost=localhost -Dhttp.proxyPort=8888
  -Djava.library.path="path-to-native-browser-libs"
  -jar kualitest-1.0b.jar "path-to-configuration-file"

```

The proxy settings are required to create web application tests. The test creator starts up an internal proxy server that intercepts all HTTP requests and saves selected requests as part of the test.

The “path-to-native-browser-libs” argument is the full path to the to the directory holding the native browser libraries

The “path-to-configuration-file” argument is the full path to the Kuali test configuration file discussed in the previous section.

### ***Test Runner Command Line***

Below is an example of the command line to start and run the test runner. It is expected that the kualitest-1.0.jar can be found on the classpath.

```

Java -Xms512m -Xmx2g org.kuali.test.runner.TestRunner
  "path-to-configuration-file"

```

The “path-to-configuration-file” argument is the full path to the kuali test configuration file discussed in the previous section.

### ***Window Specific Proxy Setup Requirements***

The Test Creator command line section above describes the proxy host and port settings required for the internal proxy to function correctly for the web tests. On most linux environments this should be all that is required. On a windows system this is not the case. In windows a proxy server must be setup through the control panel and affects all browsers. Follow the steps below to to setup a windows proxy server:

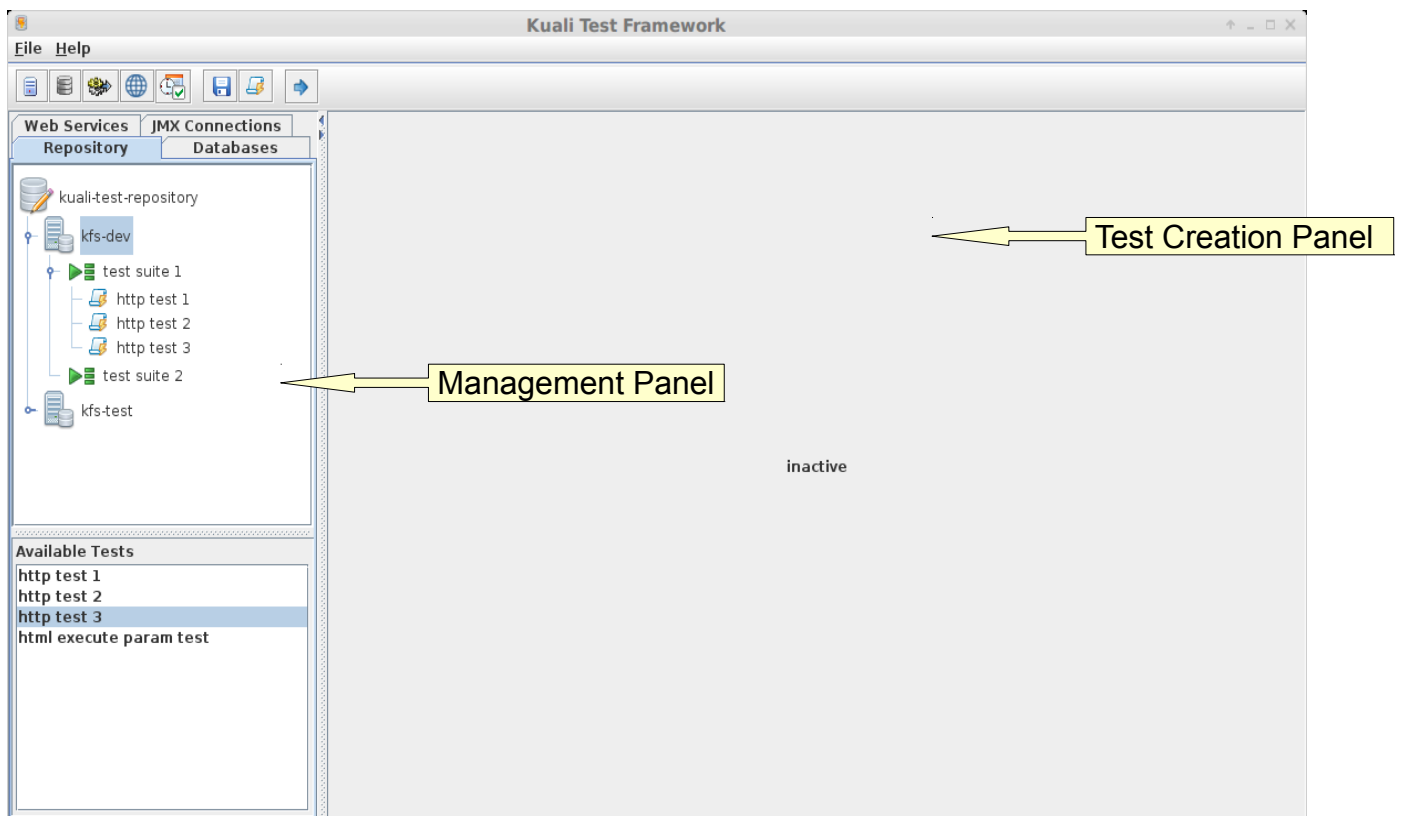
1. Open control panel .

2. Click "Network and Internet" link
3. Click "Internet Options"
4. Go to "Connections" tab
5. Click "LAN Settings"
6. Check "Use a proxy server for your LAN"
7. Enter proxy host and port used in Test Creator startup

You may also have to configure the individual browser. Most browsers have a configuration setting to specify host that cannot be a proxy – localhost is often added by default.

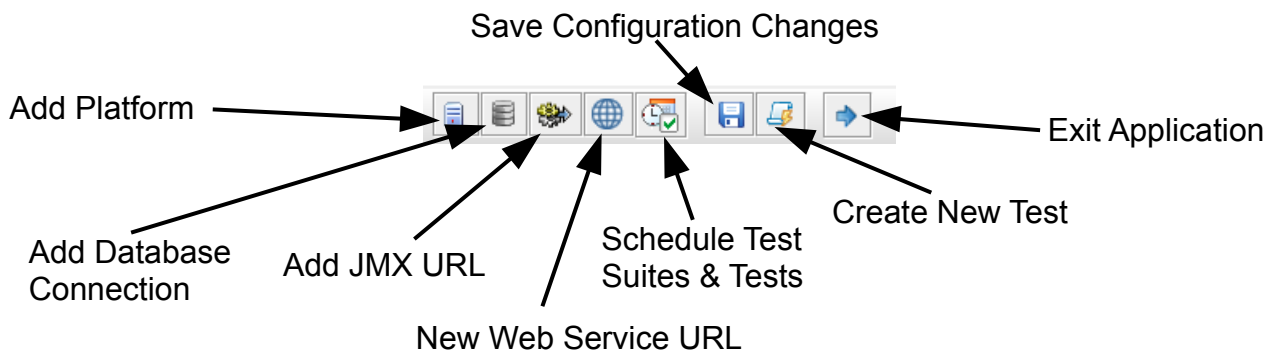
### ***Test Creator User Interface***

The Test Creator user interface is shown below. It consists of 2 main panels – a management panel and a test creation panel. The management panel provides functionality to manage the platform/test suite/test hierarchy, JDBC database connections, JMX URLs and Web Service URLs. The test creation panel provides the interface for creating web, SQL, web service and File tests.

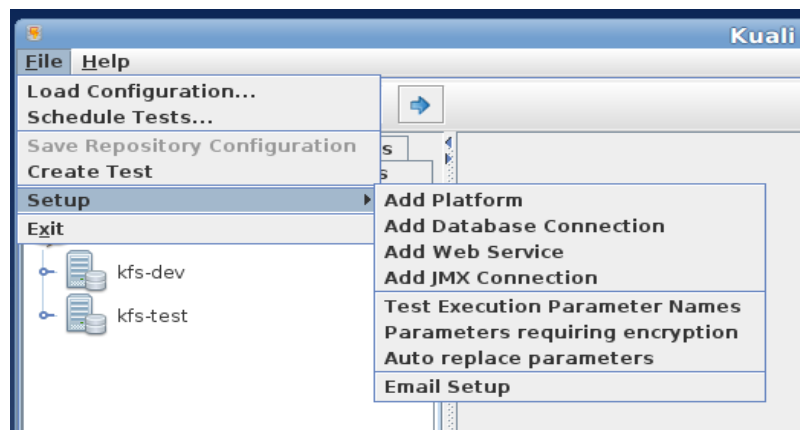


### ***Test Creator Tool Bar and Main Menu***

The test creator tool bar button descriptions are shown below:



Several main menu items parallel the tool bar button functionality described above, the other menu items are described below:



- **Test Execution Parameter Names** – provides a means of creating test execution replacement parameter names. A test execution parameter is an HTTP request parameter with the current test execution context value. For example, say I am creating a new web test that creates a requisition then pulls up several other screens referencing that new requisition. The HTTP requests saved with the test will contain the document number for the requisition. When the test is executed we do not want to reference the original requisition document number but the requisition created during the latest test run. For this scenario we can create an HTTP test execution parameter for the HTTP request parameter documentNumber and setup the test to populate the runtime HTTP requests by the requisition document number generated in the current test execution context.
- **Parameters requiring encryption** – provides a means of specifying HTTP request parameters that require encryption. During test creation selected HTTP requests are saved in the test XML document. Since the XML is stored as plain text any sensitive data in the request will be also be stored as plain text. Parameters that are specified as requiring encryption will stored in encrypted form in the test XML. For example, e request that contains the parameter “password=myspassword” would be stored in the XML test document as something similar to “password=EIXikoD3RTVn36VP3CdBuZGrWyKmIth+”



- **Auto replace parameters** – provides a means of specifying auto replacement parameters for web tests. In some test scenarios certain HTTP request parameters should always use the value from the current test execution context. Often these are hidden input fields related to authentication such as ticket values from an authentication service. HTTP request parameter names specified as auto replace will always be replaced by the current execution context values during test execution.
- **Email Setup** – provides a setup dialog for base application email parameters such as mail host, return address, default subject etc.

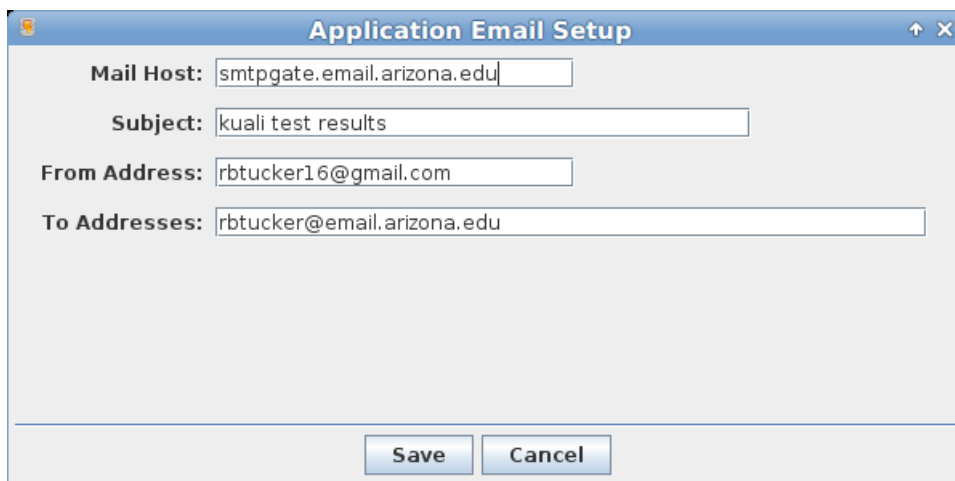
## Initial Setup

After creating the kuali test configuration XML document and getting the test creator GUI up and running and before creating tests and test suites some initial setup is required:

1. default application email parameters
2. JDBC database connections
3. JMX URLs
4. Web Service URLs
5. Platforms

## Application Email

From the application main menu click on the **File->Setup->Email Setup** menu item to display the email setup dialog:



The screenshot shows a window titled "Application Email Setup". Inside the window, there are four labeled text input fields: "Mail Host" containing "smtpgate.email.arizona.edu", "Subject" containing "kuali test results", "From Address" containing "rbtucker16@gmail.com", and "To Addresses" containing "rbtucker@email.arizona.edu". At the bottom of the window, there are two buttons: "Save" and "Cancel".

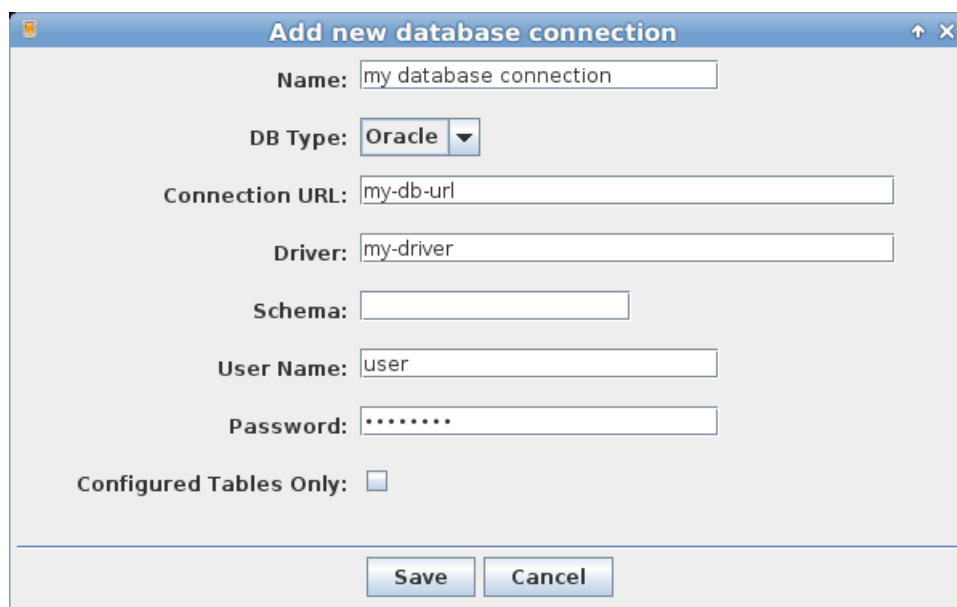
The dialog above sets the default email parameters for test result emails. The entries are for the most part self-explanatory with a couple of exceptions. The **Subject** is treated as a prefix for the test results emails – test specific information will be added to the end of this subjects line – an example is shown below:

```
kuali test results - Platform: kfs-dev, Test: http test 1
```

The **To Addresses** entry contains the default destination addresses for all tests. This field can be left empty. If not empty it can contain 1 or more addresses separated by commas that will receive test result emails for all test run.

## ***JDBC Database Connection***

From the application main menu click on the **File->Setup->Add Database Connection** menu item or the **Add Database** tool bar button to display the database connection setup dialog:



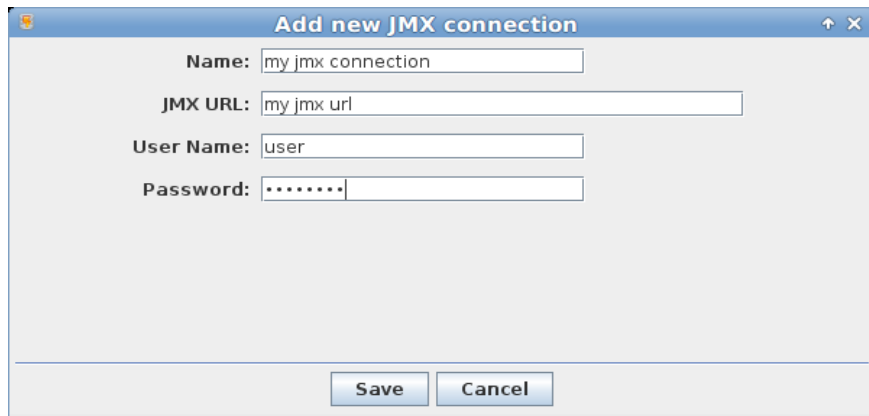
The screenshot shows a dialog box titled "Add new database connection". It contains the following fields and controls:

- Name:** A text field containing "my database connection".
- DB Type:** A dropdown menu with "Oracle" selected.
- Connection URL:** A text field containing "my-db-url".
- Driver:** A text field containing "my-driver".
- Schema:** An empty text field.
- User Name:** A text field containing "user".
- Password:** A text field containing seven dots.
- Configured Tables Only:** A checkbox that is currently unchecked.
- Buttons:** "Save" and "Cancel" buttons at the bottom right.

Enter a desired name and the JDBC parameters required to connect to the selected database. The schema field can be left blank. By default a SQL test will pull in all tables and views accessible to the database user configured for the database connection. These tables and views and their associated columns will then be presented to the user to allow for SQL query creation. The application provides a mechanism to support specifying user-friendly table and columns names to the user creating a SQL test. If the **Configured Tables Only** check box is checked only tables configured via this mechanism will display to the user. A detailed description of this functionality is found later in this section under **Additional Database Information**.

## ***JMX Connection Setup***

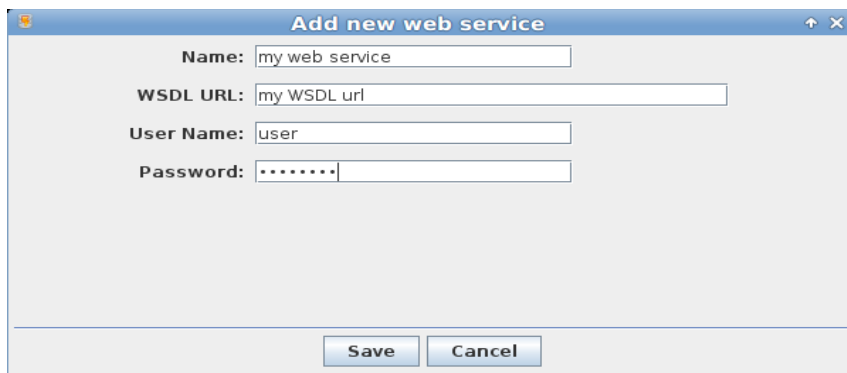
From the application main menu click on the **File->Setup->Add JMX Connection** menu item or the **Add JMX Connection** tool bar button to display the JMX connection setup dialog:



A dialog box titled "Add new JMX connection" with a blue header bar. It contains four text input fields: "Name" with the value "my jmx connection", "JMX URL" with the value "my jmx url", "User Name" with the value "user", and "Password" with masked characters ".....". At the bottom, there are two buttons: "Save" and "Cancel".

## ***Web Service Connection Setup***

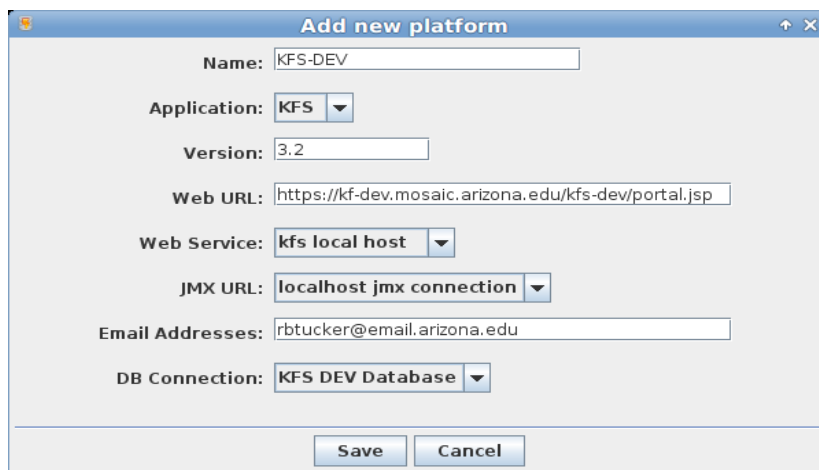
From the application main menu click on the **File->Setup->Add Web Service Connection** menu item or the **Add Web Service Connection** tool bar button to display the web service connection setup dialog:



A dialog box titled "Add new web service" with a blue header bar. It contains four text input fields: "Name" with the value "my web service", "WSDL URL" with the value "my WSDL url", "User Name" with the value "user", and "Password" with masked characters ".....". At the bottom, there are two buttons: "Save" and "Cancel".

## ***Platform Setup***

From the application main menu click on the **File->Setup->Add Platform** menu item or the **Add Platform** tool bar button to display the platform setup dialog:



A dialog box titled "Add new platform" with a blue header bar. It contains several fields: "Name" with the value "KFS-DEV", "Application" with a dropdown menu showing "KFS", "Version" with the value "3.2", "Web URL" with the value "https://kf-dev.mosaic.arizona.edu/kfs-dev/portal.jsp", "Web Service" with a dropdown menu showing "kfs local host", "JMX URL" with a dropdown menu showing "localhost jmx connection", "Email Addresses" with the value "rbtucker@email.arizona.edu", and "DB Connection" with a dropdown menu showing "KFS DEV Database". At the bottom, there are two buttons: "Save" and "Cancel".

A platform represents an application of a specific version running on a defined server. The **Web URL** entry is the web application URL. The **Web Service**, **JMX URL** and **DB Connection** entries are selected via drop down and are setup as defined in the previous sections. Once a platform is created tests and test suites can be created for the platform.

### ***Additional Database Information***

For SQL test creation the test application provides a mechanism to present user-friendly table/view and column names to the end user. In addition, pseudo foreign key relationships can be defined. For example, in Kuali many KC and KFS tables reference document numbers that also reside in RICE tables but there are no foreign key constraints in the database. Since the SQL query creation panel of the test creator uses foreign keys to logically display table relationships the associated KFS/KC/RICE table relationships are not displayed; however, by configuring pseudo foreign keys the UI can display these relationships.

To define additional database information create an XML file in the directory specified in the Kuali test framework configuration under the tag shown below:

```
<additional-db-info-location>path-to-db-info</additional-db-info-location>
```

The file naming convention must be as follows:

```
[application-name]-additional-db-info.xml
```

for example kfs-additional-db-info.xml.

Example XML is shown below:

```
<additional-database-info xmlns="test.kuali.org">
  <application>
    <application-name>KFS</application-name>
    <tables>
      <table>
        <table-name>FP_DV_EXT_T</table-name>
        <display-name>
          DisbursementVoucherDocumentExtension
        </display-name>
        <columns>
          <column>
            <column-name>FDOC_NBR</column-name>
            <display-name>documentNumber</display-name>
          </column>
          <column>
            <column-name>VER_NBR</column-name>
            <display-name>versionNumber</display-name>
          </column>
          <column>
            <column-name>OBJ_ID</column-name>
            <display-name>objectId</display-name>
          </column>
          <column>
            <column-name>BATCH_ID</column-name>
            <display-name>batchId</display-name>
          </column>
        </columns>
      </table>
    </tables>
  </application>
</additional-database-info>
```

```

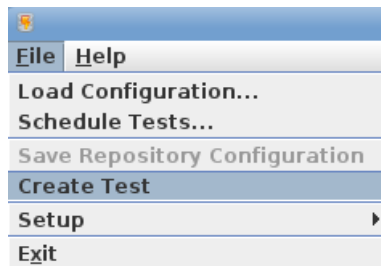
</columns>
<custom-foreign-keys>
  <custom-foreign-key>
    <name>customfkl</name>
    <primary-table-name>
      KREW_DOC_HDR_T
    </primary-table-name>
    <foreign-key-column-pair>
      <primary-column>
        DOC_HDR_ID
      </primary-column>
      <primary-column-type>
        number
      </primary-column-type>
      <foreign-column>
        FDOC_NBR
      </foreign-column>
      <foreign-column-type>
        string
      </foreign-column-type>
    </foreign-key-column-pair>
  </custom-foreign-key>
</custom-foreign-keys>
</table>

```

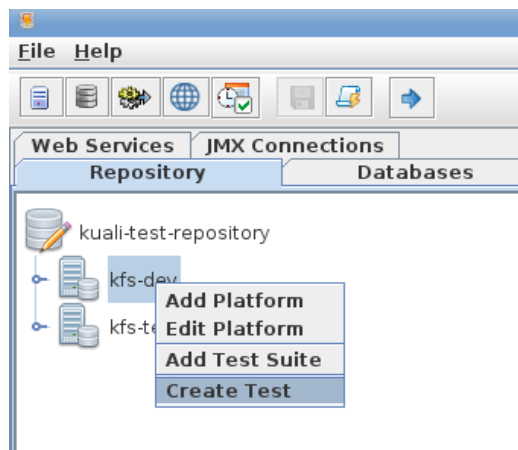
A complete description is found in **Appendix B**.

## Creating a New Test

To create a new test, select **File->Create Test** from the main application menu,



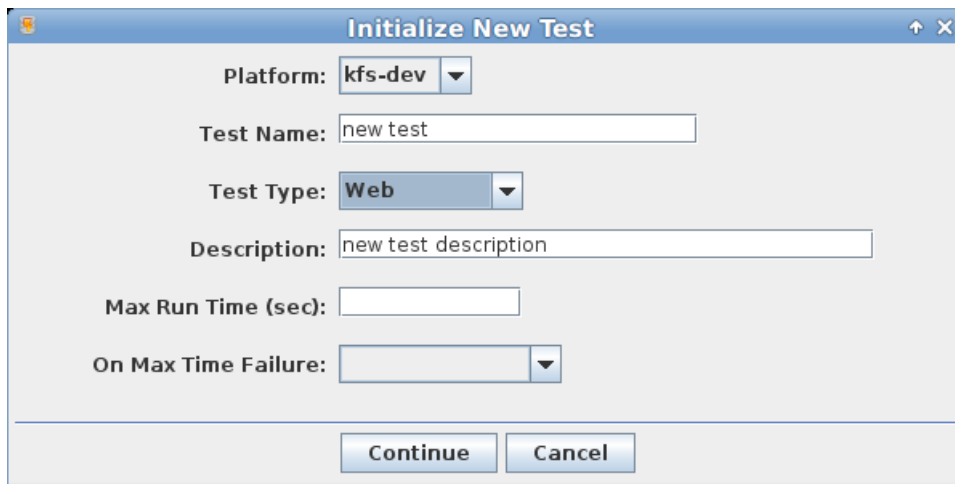
or right click on the desired platform on the repository tab,



or click the **create new test** button on the application tool bar,



to display the **Initialize New Test** dialog:

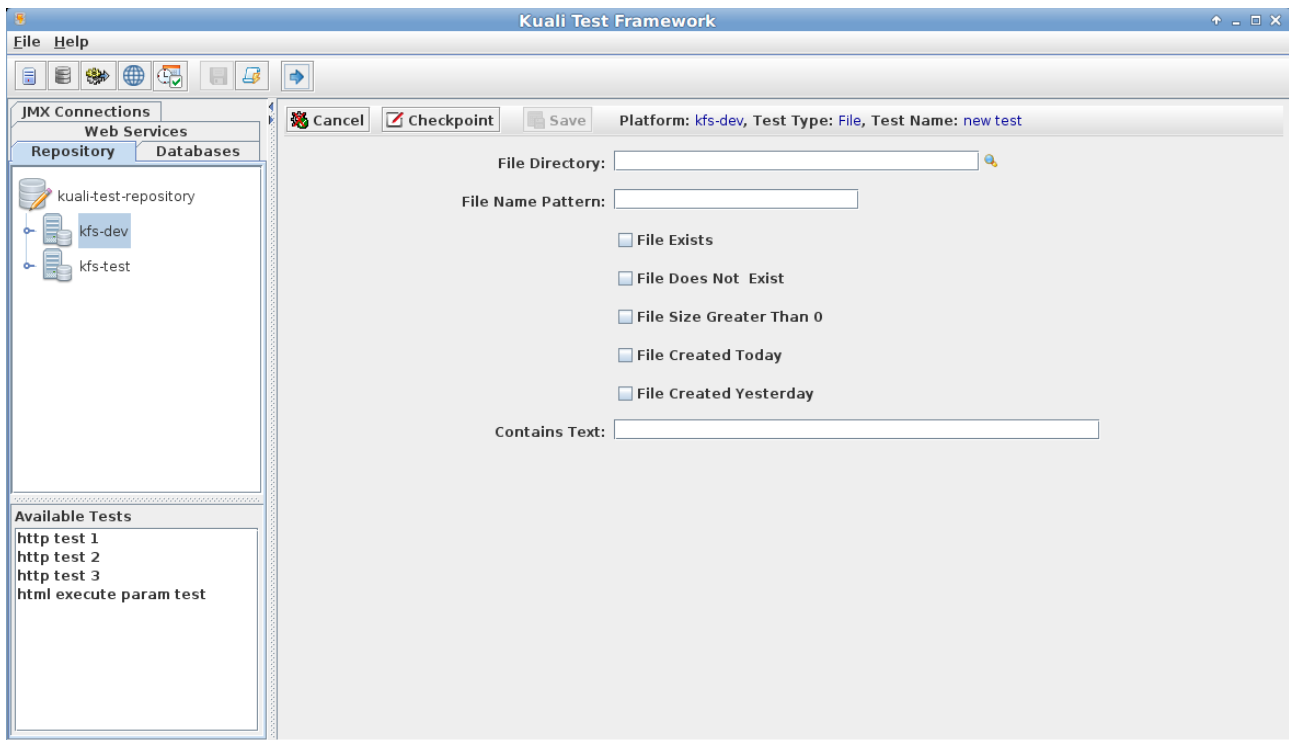


Below are the descriptions of the dialog entry fields:

- **Platform** – the platform that this test will be associated with.
- **Test Name** – desired test name – this must be unique for the platform.
- **Test Type** – test type from drop down selection – one of 4 choices: database, file web or web service.
- **Description** – test description
- **Max Run Time (sec)** – the maximum time in seconds to allow this test to run before flagging an error. The field is not required.
- **On Max Time Failure** – if the max run time is set, select a failure status from drop down selection to associate with the error – one of 4 choices: ignore, warning, error – continue or error – halt test.

## Creating a File Test

Pull up the **Initialize New Test** dialog as described above and complete the entries as desired. Ensure that the **Test Type** of **File** is selected from the drop down. Click **Continue** to load the file test setup panel.



Complete the entry fields on the file test panel and click the checkpoint button



to display the **Add new checkpoint** dialog and save the checkpoint. Click the save button



to save the test.

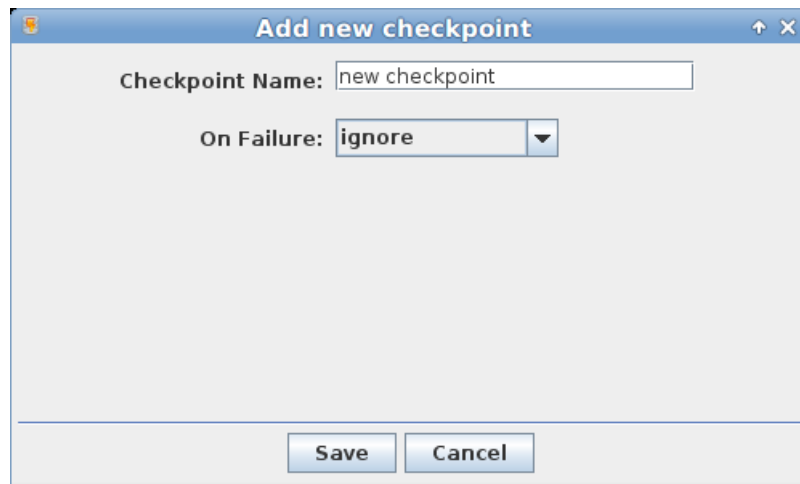
The File test panel entry fields are described below:

- **File Directory** – the directory where the files of interest reside.
- **File Name Pattern** – the name pattern of file to look for. An asterisk (\*) can be used as a wild card character.
- **File Exists** - check box to indicate test that file exists
- **File Does Not Exist** – check box to indicate test that file does not exist
- **File Size Greater Than 0** – check box to indicate a test that files exists with size greater than zero.
- **File Created Today** – check box to indicate a test that file was created today
- **File Created Yesterday** – check box to indicate a test that file was created yesterday
- **Contains Text** – File will be tested to see if the entered text is found in the file.

Various combinations of these test properties are allowed.

## ***Completing Checkpoint Dialog***

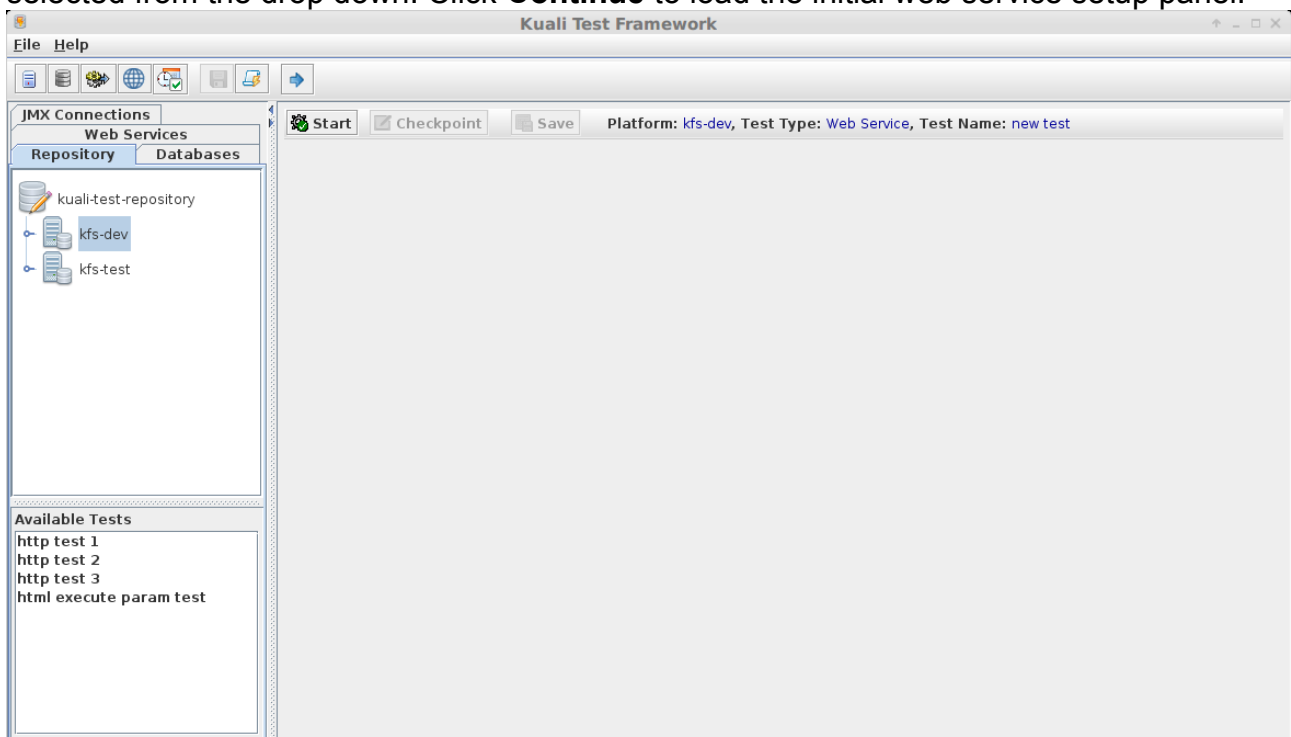
When the checkpoint button is clicked the **Add new checkpoint** dialog will display:



Enter a valid check point name and select a failure status to associate with a file test failure.

## Creating a Web Service Test

Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Web Service** is selected from the drop down. Click **Continue** to load the initial web service setup panel.

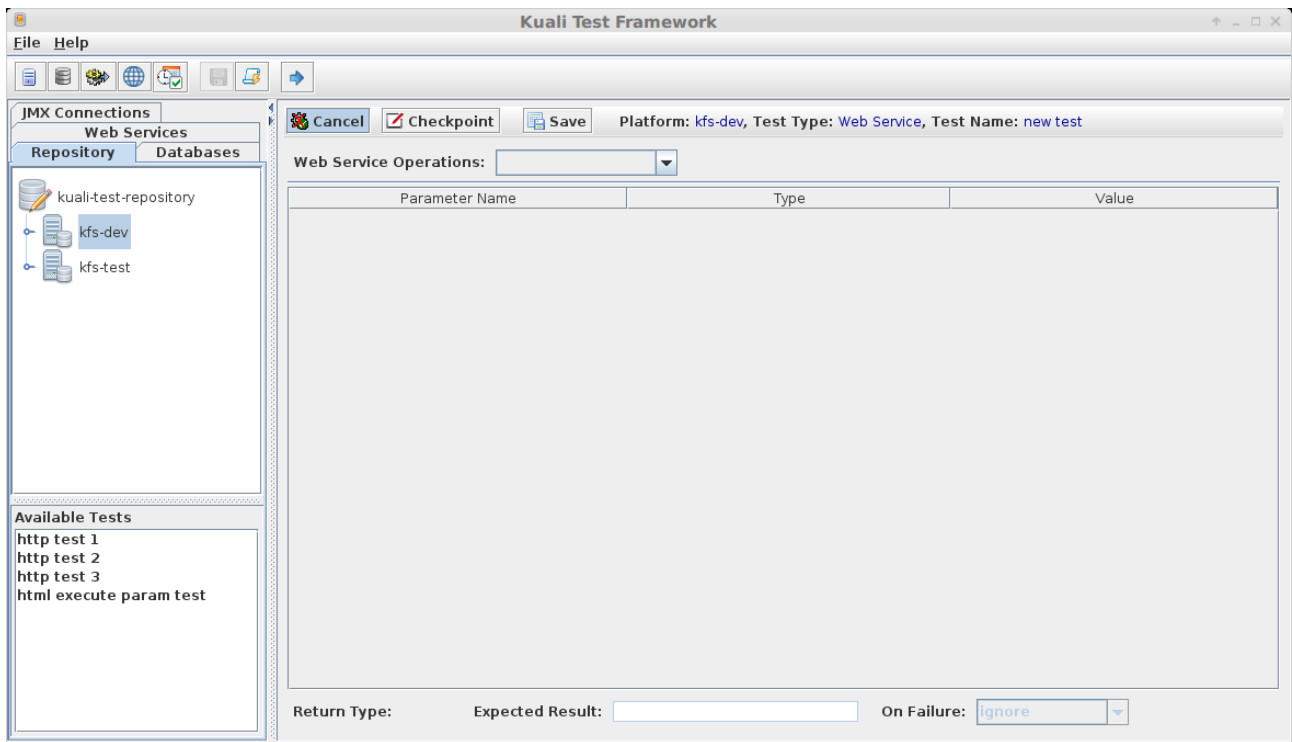


Click the Start button to connect to the web service setup for the platform.

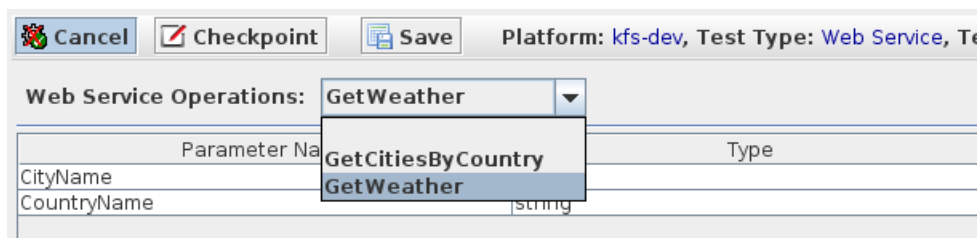


If the connection succeeds the full web service test setup panel will display.

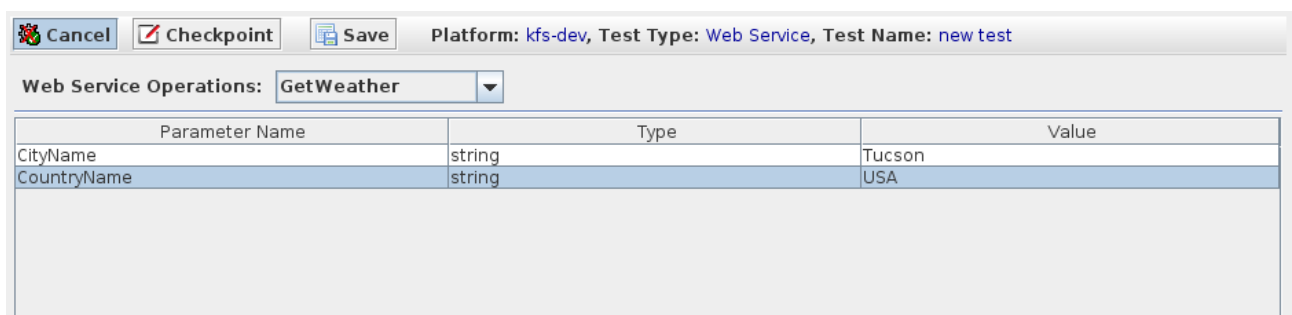




From the **Web Service Operations** drop down select a desired operation for a checkpoint.



Complete the input parameter entries:



And the expected result field:



Select the failure status to set if expected result is not retrieved and click the checkpoint button to display the **Add new checkpoint** dialog.

A dialog box titled "Add new checkpoint" with a blue header bar. It contains three input fields: "Checkpoint Name:" with the text "new checkpoint", "Max Run Time (sec):" which is empty, and "On Failure:" with a dropdown menu showing "ignore". At the bottom are "Save" and "Cancel" buttons.

**Add new checkpoint**

Checkpoint Name: new checkpoint

Max Run Time (sec):

On Failure: ignore

Save Cancel

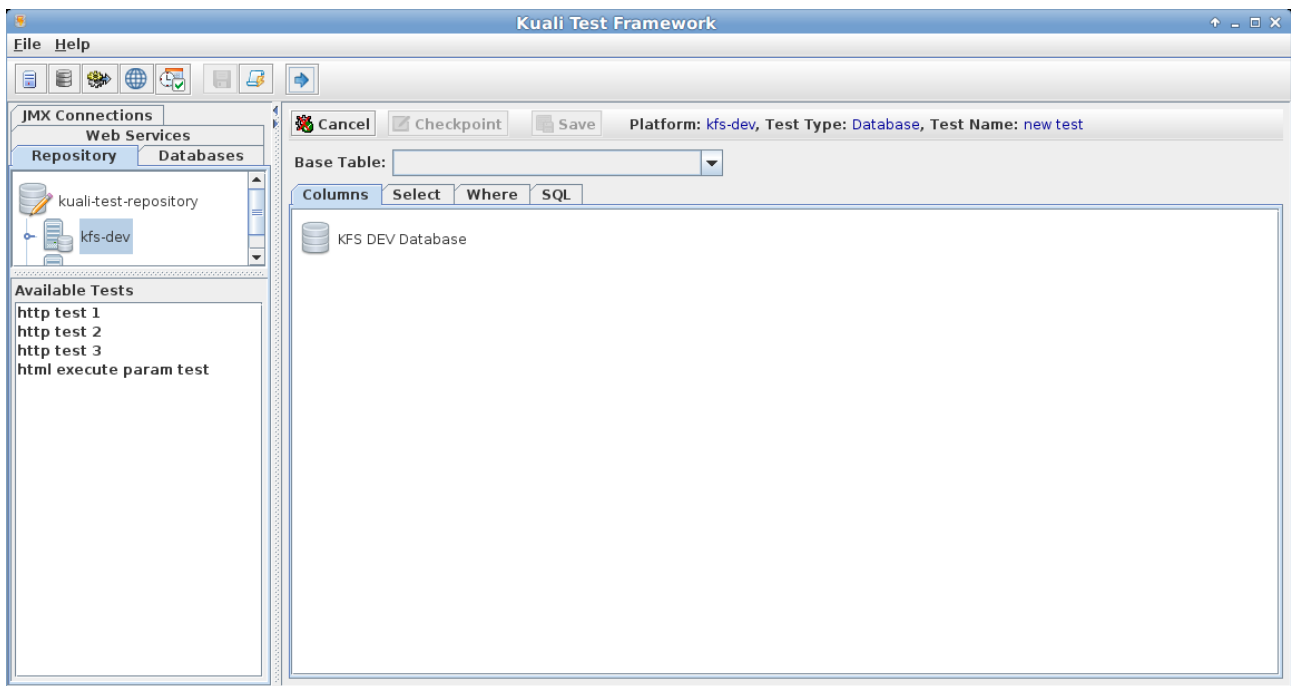
Enter **Checkpoint Name**, a Max Run Time and a max runtime failure status if desired. Click the **Save** button to save the check point.



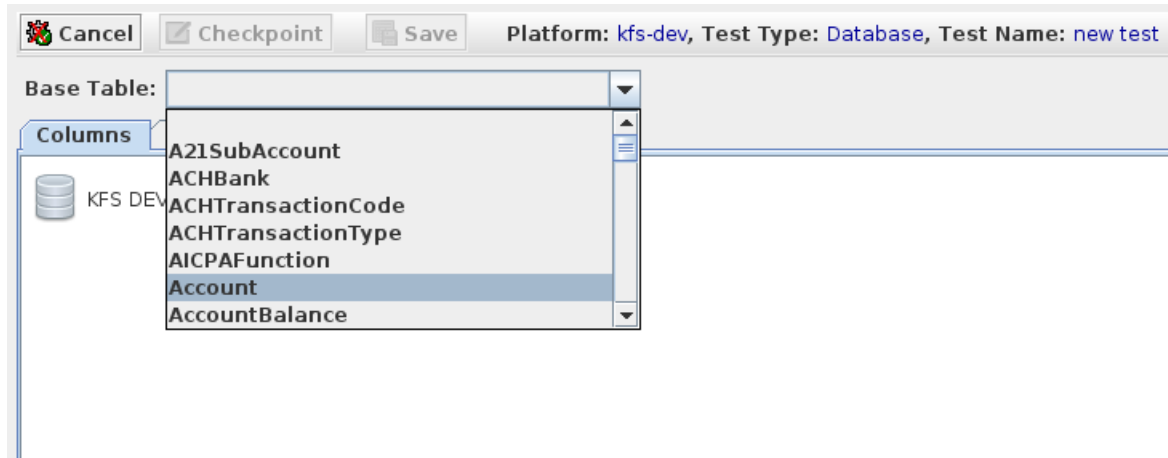
to save the test.

## Creating a Database Test

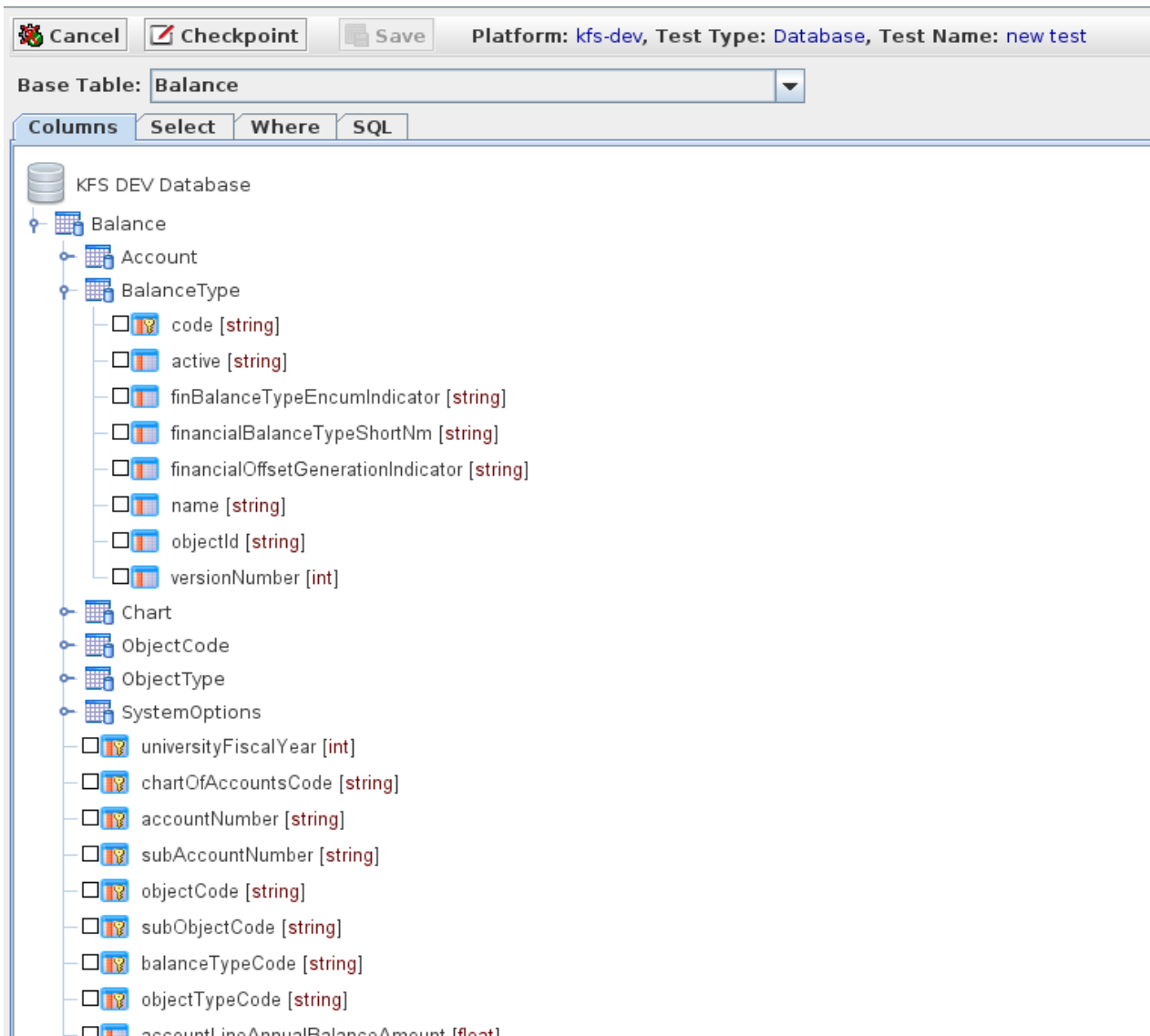
Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Database** is selected from the drop down. Click **Continue** to load the initial database test setup panel.



To begin the process of creating a SQL query for the database test, select a base table from the **Base Table** drop down.



Once selected the SQL query creation tabs will display.



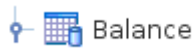
## Selecting Columns for Database Query

The first step in creating a SQL query is to select the desired tables/columns you want to work with. The **Columns** tab of the SQL query panel provides a tree display of related tables with the **Base Table** selection as the root. Table relationships are displayed based on database foreign key constraints. You can also create pseudo foreign key relationships as described in section **Additional Database Information**. Description of the various column selection tree elements are described below:

### Database associated with test platform



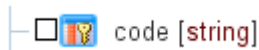
## Database Table



## Table Column

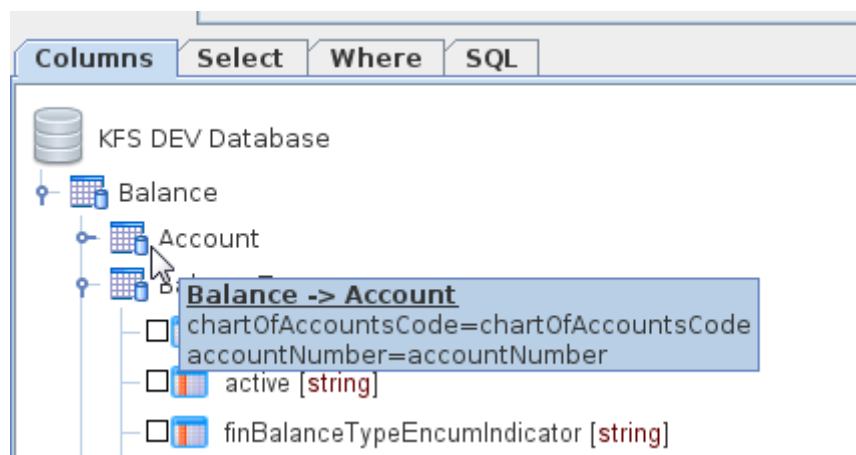


## Primary Key Table Column

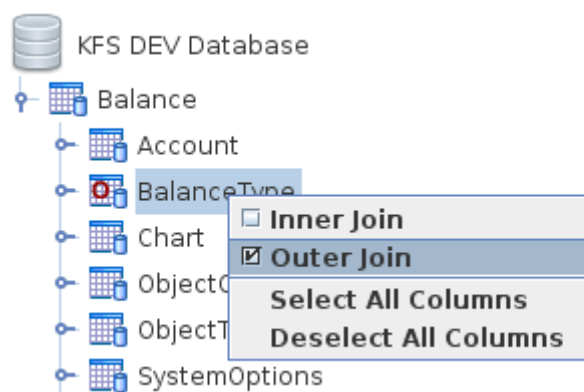


Column data type is displayed in square brackets: [string]

As mentioned earlier, the table hierarchy is based on foreign key constraints with the selected base table as root. To display the foreign key information hover the cursor over the table of interest:

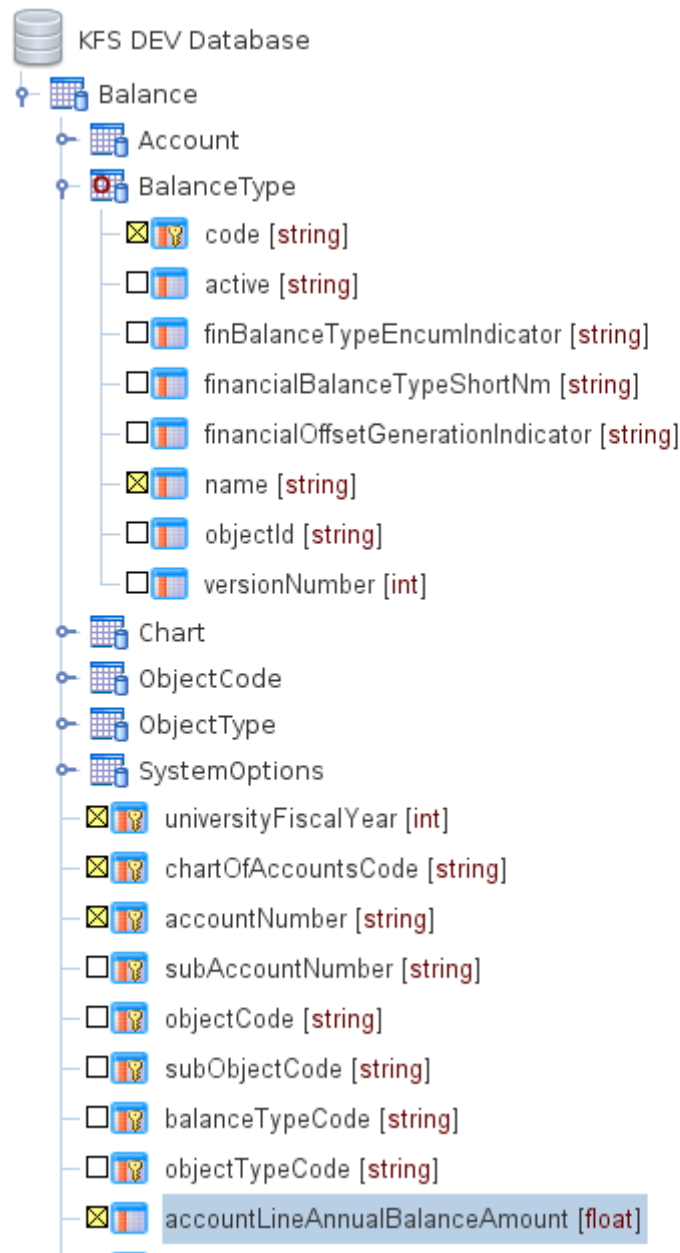


By default multi-table column selections will create an inner-join query. If an outer join is desired, right click on the table desired and select the outer join option:



The table will display with a red “O” to indicate that this relationship will generate an outer join.

Click the check box next to all columns that you wish to include in the SQL query. The selected columns will be the only columns that can be used in the select, where and order by clauses of the generated query.



### ***Building SQL Select and Order By Clause***

Click the **Select** tab on the SQL query creation panel to build the select clause:

Cancel
 Checkpoint
 Save
 Platform: kfs-dev, Test Type: Database, Test Name: new test

Base Table: Balance

Columns **Select** Where SQL

Select Columns

Add Column
 Delete Column
 ☐ DISTINCT

table	column	function	order	asc/desc
-------	--------	----------	-------	----------

To add a new select column click the Add Column button:



Columns **Select** Where SQL

Select Columns

Add Column
 Delete Column
 ☐ DISTINCT

table	column	function	order	asc/desc
<span>Balance</span>				
<span>BalanceType</span>				

Choose the desired table and column from the drop down selections. To apply an aggregate function to a column choose the desired function from the function drop down.

Columns **Select** Where SQL

Select Columns

Add Column
 Delete Column
 ☐ DISTINCT

table	column	function	order	asc/desc
<span>Balance</span>	<span>accountLineAnnualBalanceAmount [float]</span>	AVG COUNT MAX MIN <b>SUM</b>		

The application will handle generating the correct group by clause if aggregate functions are selected.

Enter an integer order position in the order column if desired to generate an order by clause. If an order position is specified **asc/desc** can be selected via drop down. If neither ascending or descending is specified the generated order by clause will default to ascending for the column.

Columns					Select	Where	SQL
Select Columns							
+ Add Column		X Delete Column		<input type="checkbox"/> DISTINCT			
table	column	function	order	asc/desc			
Balance	accountLineAnnualBalanceAmount	float	1	<div> <div>ASC</div> <div>DESC</div> </div>			

To generate a distinct SQL query check the **DISTINCT** check box.

Select Columns		
+ Add Column	X Delete Column	<input checked="" type="checkbox"/> DISTINCT

## Building SQL Where Clause

Click the **Where** tab on the SQL query creation panel to build the where clause:

<div> <div>Cancel</div> <div>Checkpoint</div> <div>Save</div> </div> Platform: kfs-dev, Test Type: Database, Test Name: new test						
Base Table: Balance						
Columns Select Where SQL						
Where Comparisons						
+ Add Comparison		X Delete Comparison				
and/or	(	table	column	operator	value	)
<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>						

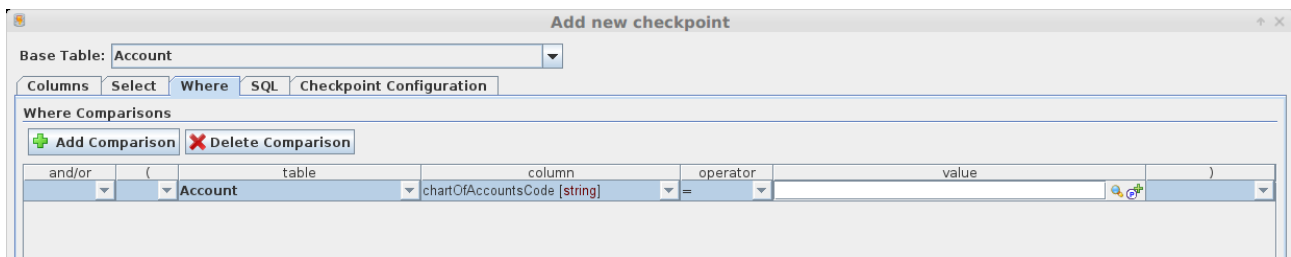
To add a new where comparison click the **Add Comparison** button

+ Add Comparison

Columns Select Where SQL						
Where Comparisons						
+ Add Comparison		X Delete Comparison				
and/or	(	table	column	operator	value	)
		Balance		=		
		BalanceType				
		Balance -> BalanceType balanceTypeCode=code				

Select the desired table, column and operator then enter an appropriate comparison value.

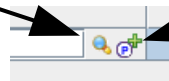




Notice the icons on the value entry column:

Show Value Lookup

Select Test Execution Parameter



The Test Execution Parameter functionality is similar to what is described in section **Creating Web Test Execution Parameter**. In an HTTP SQL checkpoint the where comparison value can be pulled from the current test execution context. This icon will not display in a SQL test.

The Show Lookup Value can display a list of available values for a where comparison entry. The lookup values are pulled from the database and are associated with column selected columns. This association is configured in the additional database information XML. A full description of the setup is found in Appendix B. There are ways to create the lookup association - Examples are shown below:

### Global lookup by column name:

```
<lookups>
  <lookup>
    <column-name>FIN_COA_CD</column-name>
    <sql>select fin_coa_cd, fin_coa_cd from ca_chart_t order by 2</sql>
  </lookup>
</lookups>
```

with this setup the lookup icon and lookup functionality will be available on all columns of name FIN\_COA\_CD

### Column specific lookup

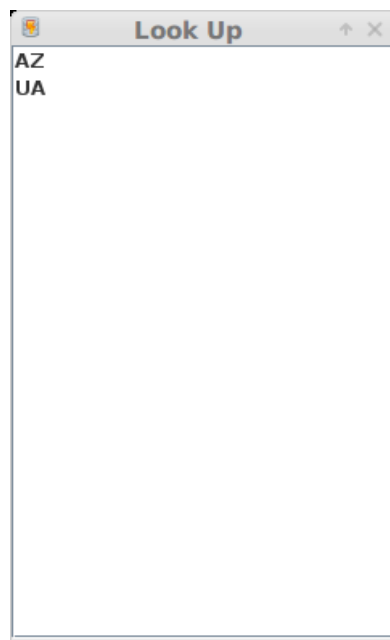
Lookup functionality can also be applied to an individual column definition. If there is both a global lookup and column-specific lookup the column-specific lookup will take precedence. Example XML configuration is shown below:

```
<column>
  <column-name>CR_ID</column-name>
  <display-name>id</display-name>
  <lookup-sql-select>
    select fin_coa_cd, fin_coa_cd from ca_chart_t order by 2
  </lookup-sql-select>
</column>
```

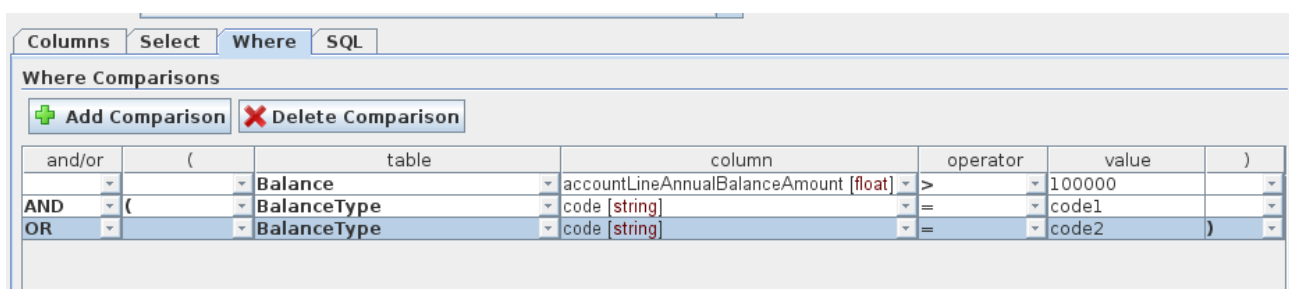
The lookup select should be of form

```
select key-value, display-value from table where active = 'Y' order by 2
```

The display and key values can be the same. The lookup functionality is designed to display limited lists of values with a single key definition. The key value will be set as the where value lookup. The lookup value selection dialog is shown below:\



If more than 1 comparison is entered the **and/or** values and parenthesis values - ( ) - can be selected to further define the where clause.



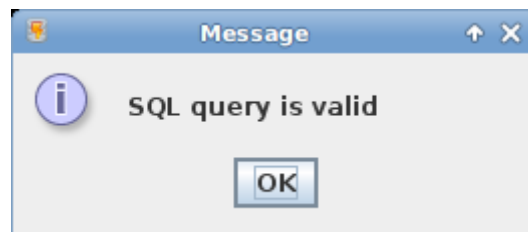
## ***Displaying Generated SQL and Verifying Validity***

To view and validate the SQL statement that will be generated, click the **SQL** tab.

Columns	Select	Where	SQL
<div>Check Generated SQL</div> <pre>select distinct t1.ACLN_ANNL_BAL_AMT, t2.FIN_BALANCE_TYP_NM from GL_BALANCE_T t1 left outer join CA_BALANCE_TYPE_T t2 on ( t2.FIN_BALANCE_TYP_CD = t1.FIN_BALANCE_TYP_CD) where t1.ACLN_ANNL_BAL_AMT &gt; 100000 AND ( t2.FIN_BALANCE_TYP_CD = 'code1' OR t2.FIN_BALANCE_TYP_CD = 'code2') order by t1.ACLN_ANNL_BAL_AMT</pre>			

Click the **Check Generated SQL** button to verify the SQL.

Check Generated SQL



Right click on the SQL display screen and click the **Copy SQL** popup menu item to copy the generated SQL to the clipboard.

ColumnsSelectWhereSQL

Check Generated SQL

```
select
  distinct
    t1.ACLN_ANNL_BAL_AMT,
    t2.FIN_BALANCE_TYP_NM
  from
    GL_BALANCE_T t1
    left outer join CA_BALANCE_TYPE_T t2 on (
      t2.FIN_BALANCE_TYP_CD = t1.FIN_BALANCE_TYP_CD)
  where
    t1.ACLN_ANNL_BAL_AMT > 100000
    AND ( t2.FIN_BALANCE_TYP_CD = 'code1'
    OR t2.FIN_BALANCE_TYP_CD = 'code2')
  order by
    t1.ACLN_ANNL_BAL_AMT
```

Copy SQL

### Creating SQL Checkpoint

Once the SQL query is created checkpoints can be applied. Click the **Checkpoint** button



to display the **Add new checkpoint** dialog:

Add new checkpoint

Checkpoint Name:

new checkpoint

Checkpoint Property:

single row exists

On Failure:

ignore

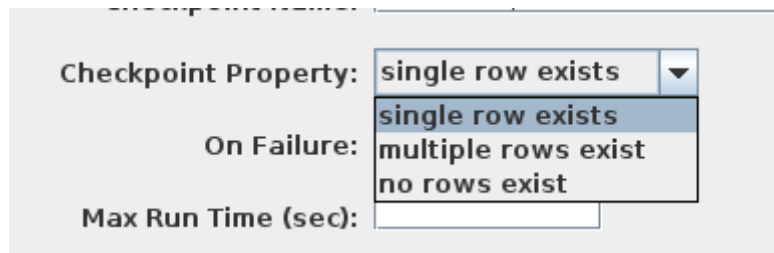
Max Run Time (sec):

☐ Save SQL Query Results

Save

Cancel

Assign an appropriate name to this check point and select the desired **Checkpoint Property** for test evaluation purposes.

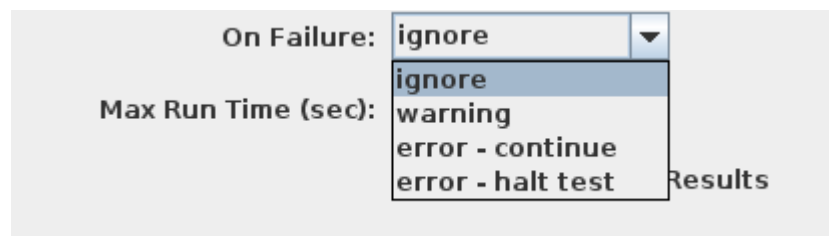


Checkpoint Property: single row exists ▼

On Failure: single row exists  
multiple rows exist  
no rows exist

Max Run Time (sec):

Select the appropriate On Failure status to set in the event of check point failure.



On Failure: ignore ▼

Max Run Time (sec):

Results

If desired a maximum query run time can be entered. If the query execution exceeds this time then the check point fails.

To save query results generated from a SQL check point query check the **Save SQL Query Results** check box. The query results will be saved as a comma-delimited file and delivered as an email attachment with the test results email.

Click the save button to close the dialog and save the check point.

To save the SQL test click the test Save button:



## Creating a Web Application Test

Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Web** is selected from the drop down. Click **Continue** to load the initial database test setup panel.

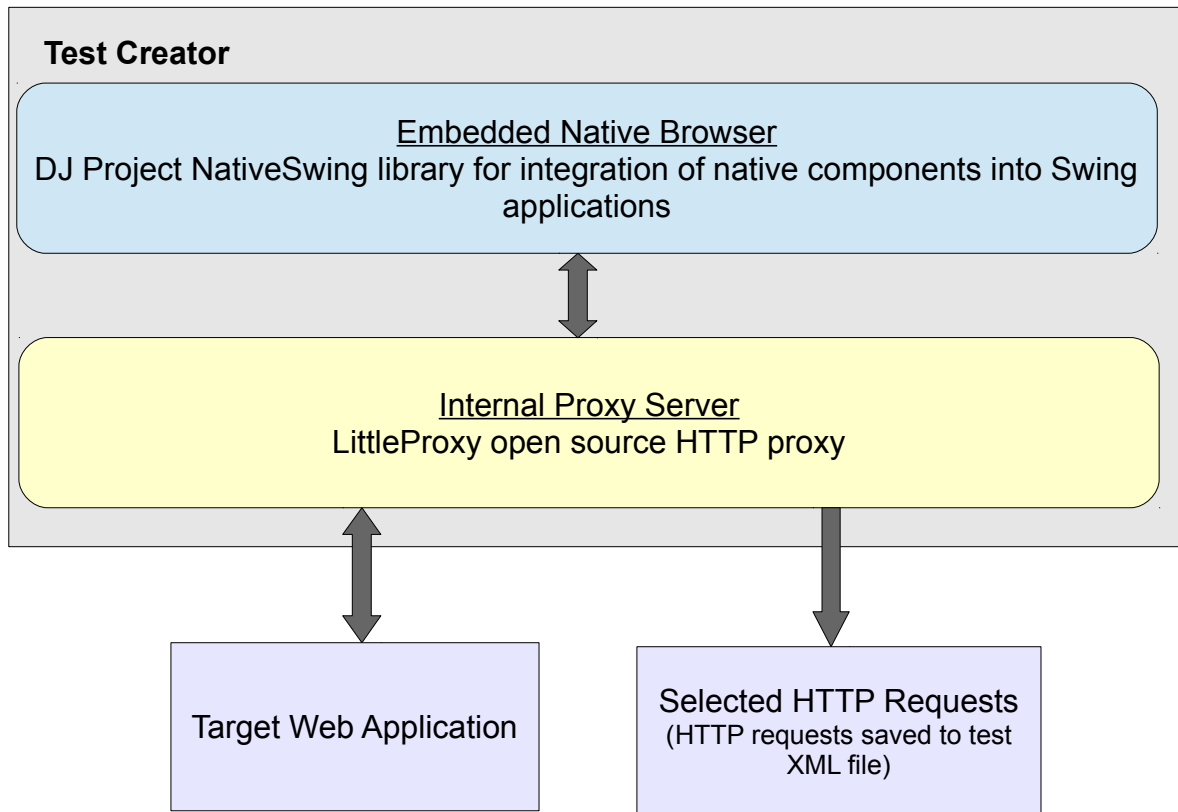
### ***Web Application Test Framework Overview***

The web application test framework is relatively complex and requires a bit of an overview.

#### **Test Creator**

The test creator is a Java Swing application used to manage the test framework and

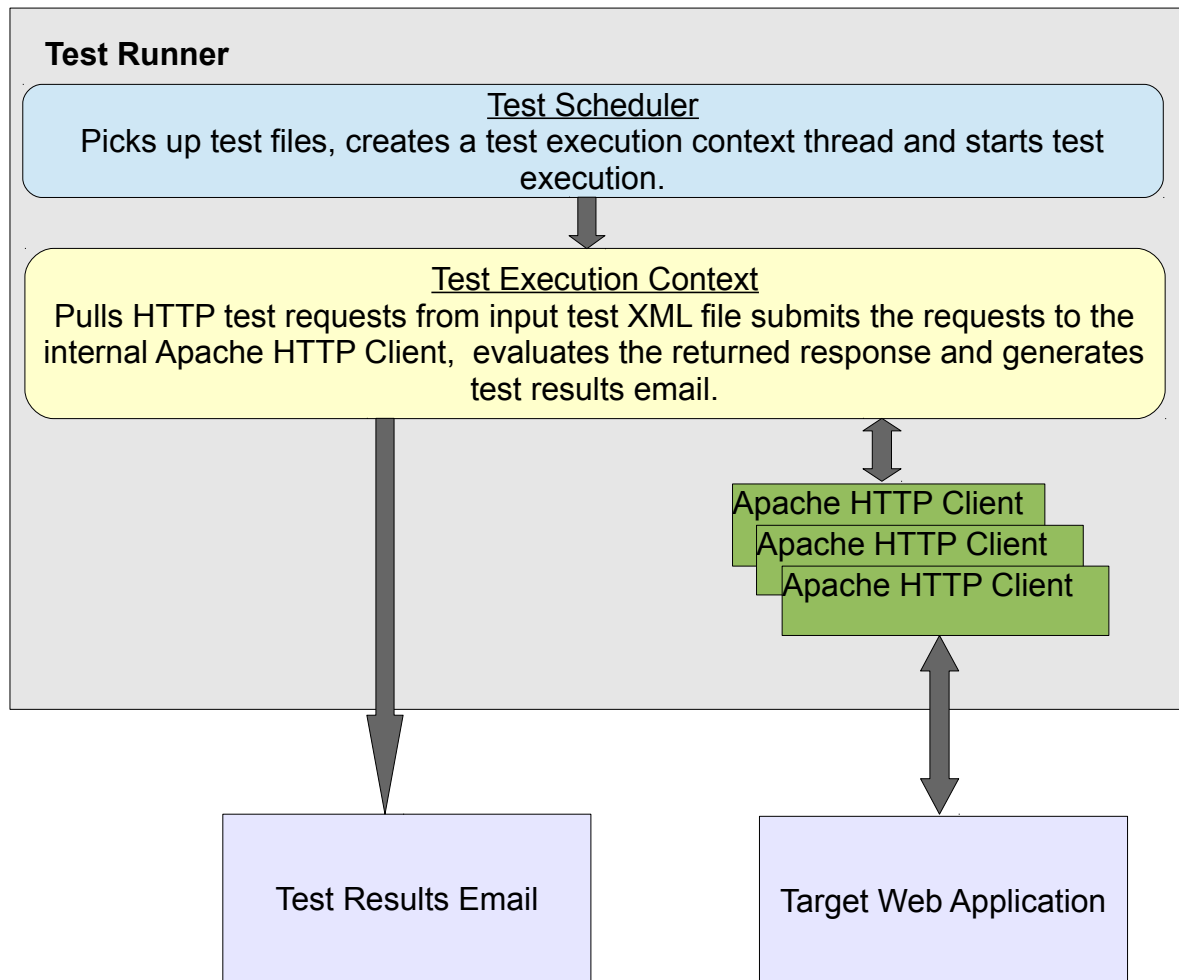
create tests. For web application tests the test creator uses an embedded native browser instance and an internal proxy server to facilitate test creation as shown in the diagram below:



The proxy server allows the application to act as a “man-in-the-middle” and grab non encrypted HTTPS requests before they are sent to the target web application. To use the proxy server the test creator must be started with the appropriate input parameters as described in section ***Test Creator Command Line***.

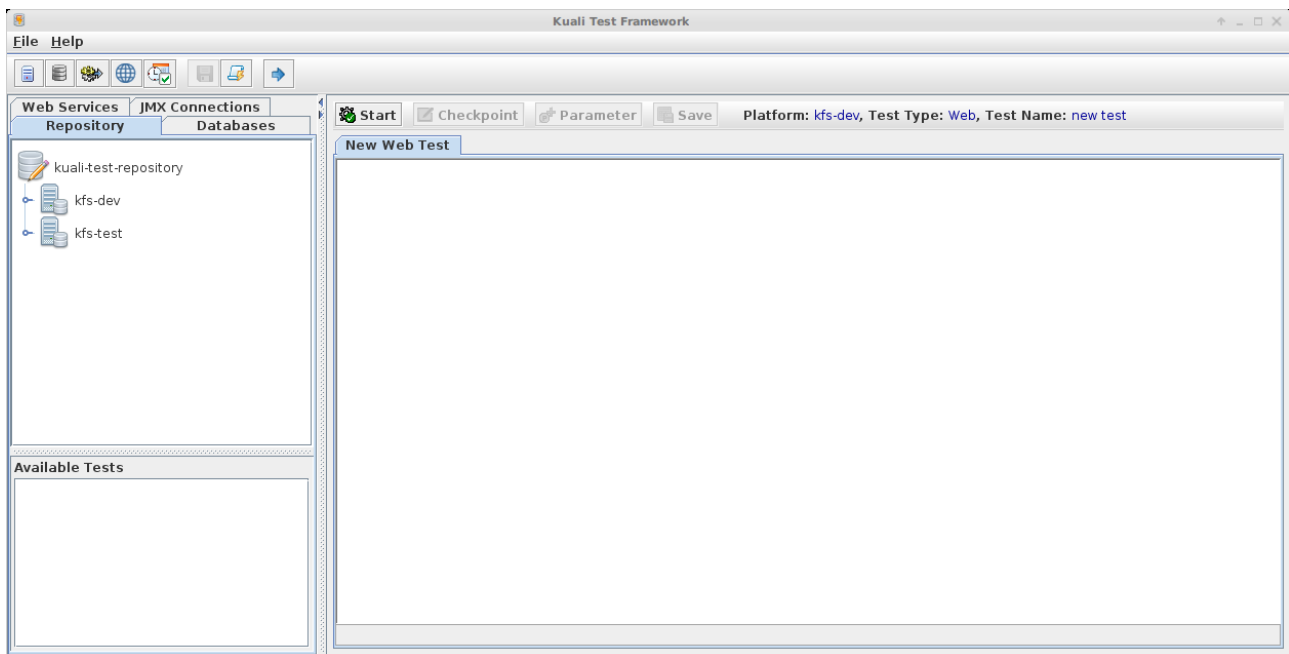
### Test Runner

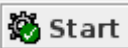
The test runner is a multi-threaded Java application that executes scheduled tests. For web application testing the test runner executes HTTP requests from a test file using the Apache Http Client libraries as shown in the diagram below:



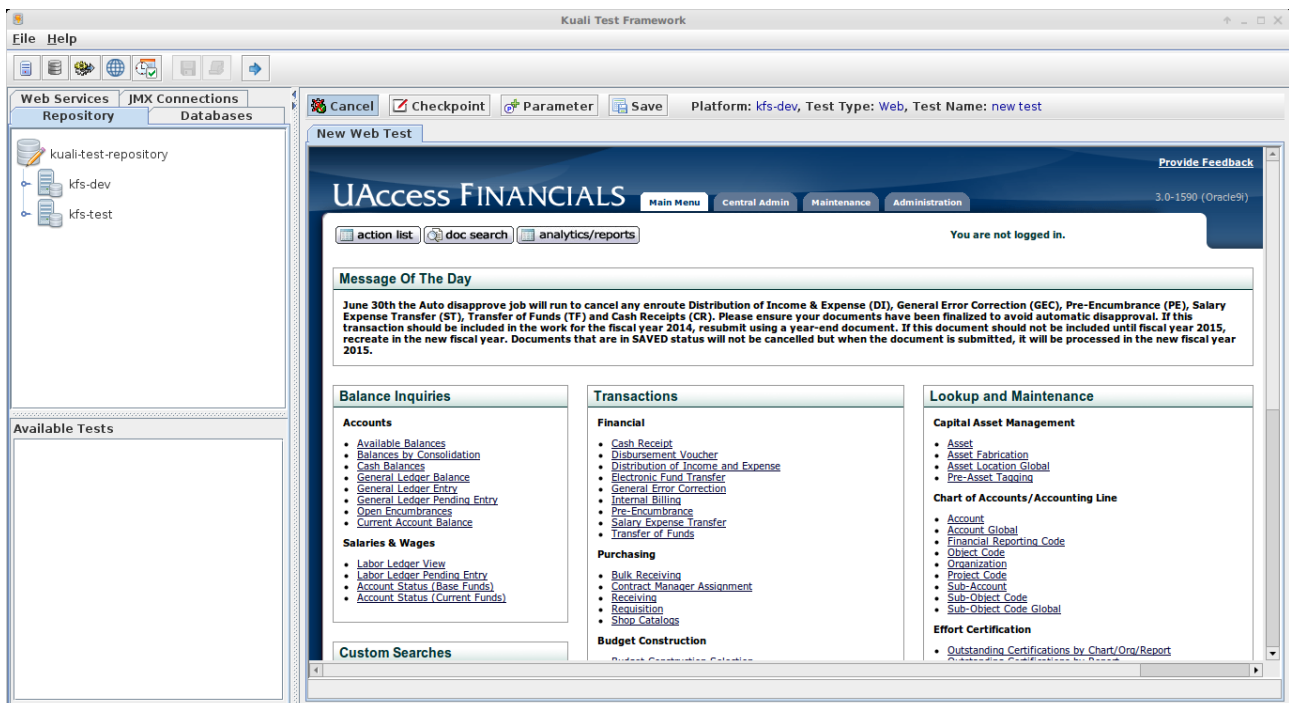
### ***Starting a Web Application Test***

After clicking the **Continue** button on the **Initialize New Test** dialog the web test panel should display.

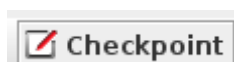


Click the Start button  to begin the test.

Once the web test is initialized the default native browser should display in the web test panel:

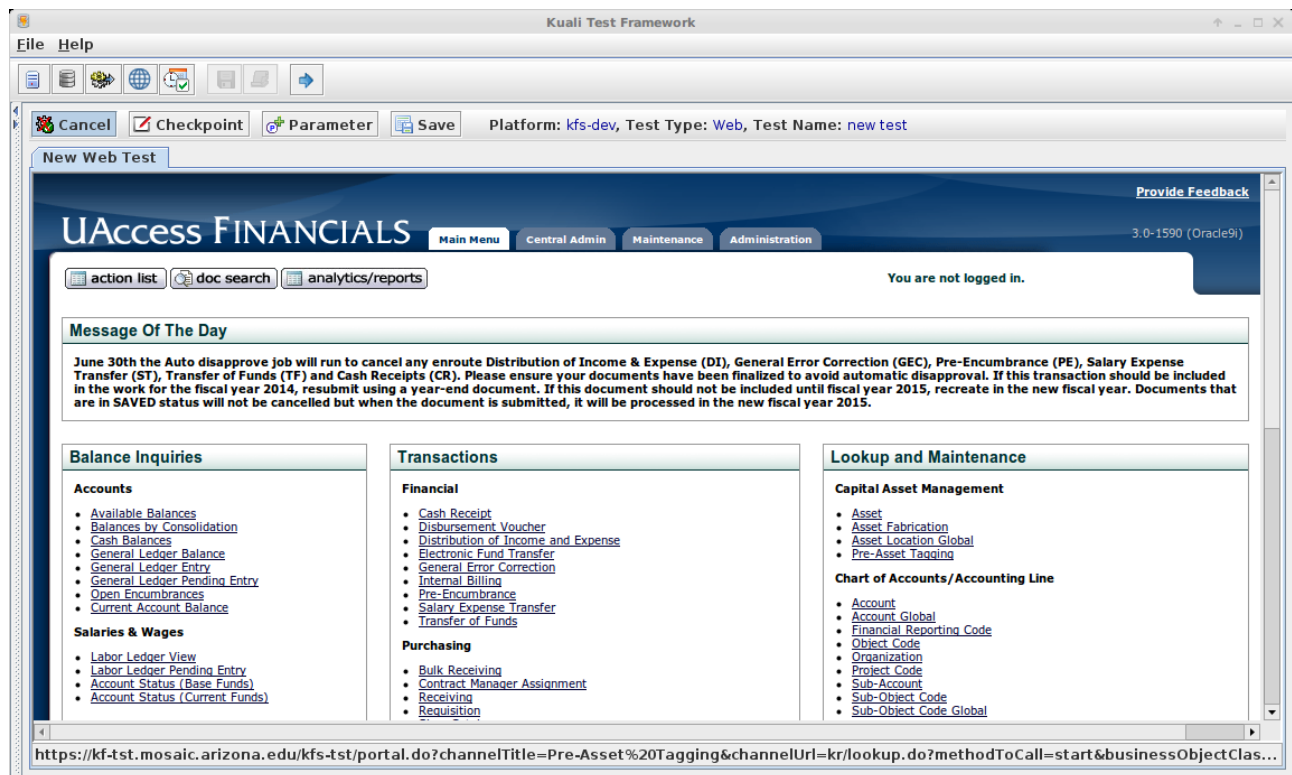
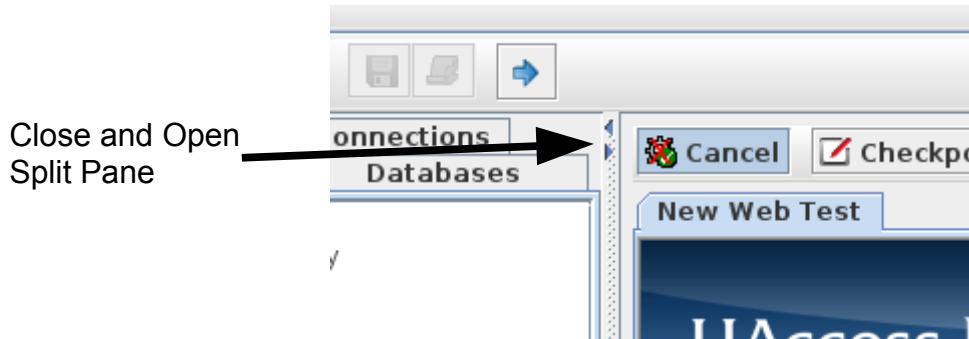


Now you can navigate through the target web application as normal. At any point you can click the Checkpoint button

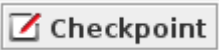


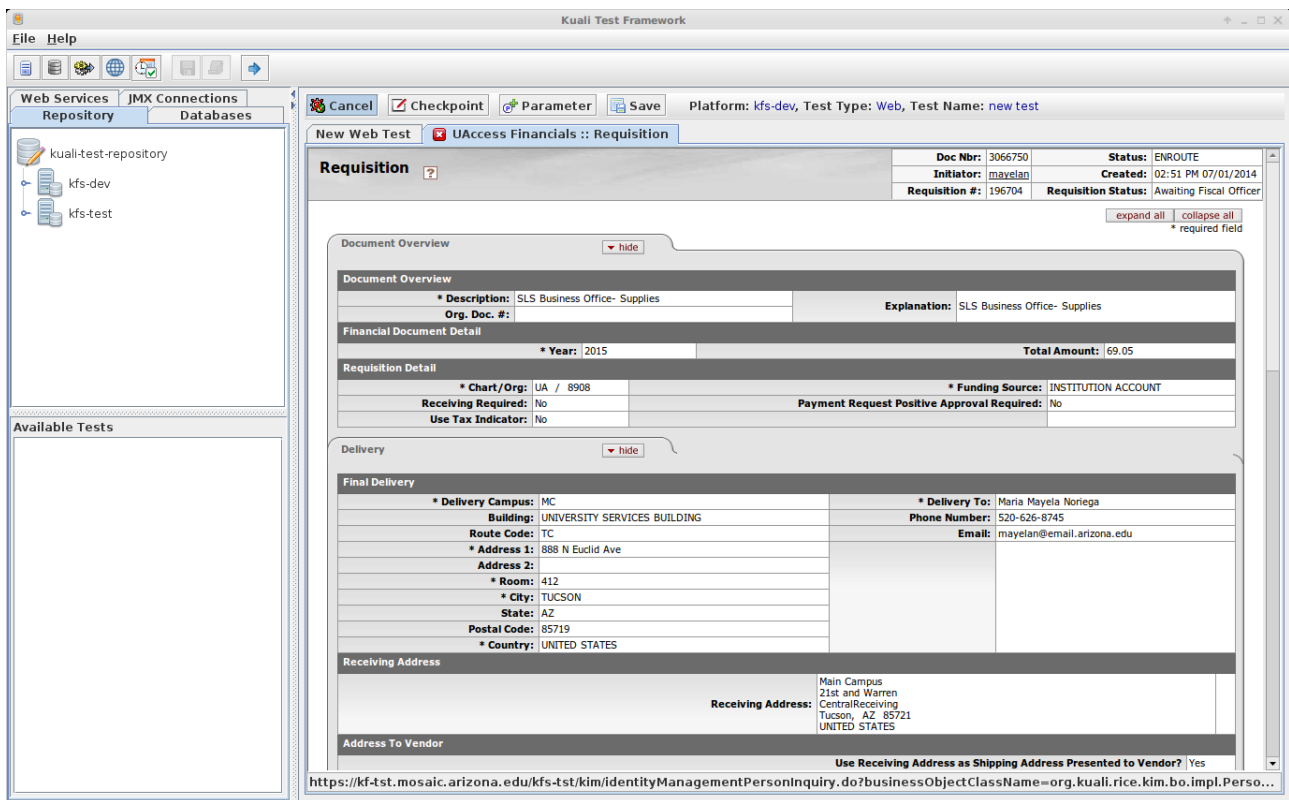


to create a check point that will be used as evaluation criteria during test runs. To better see the full web page click the split pane icon shown below – this will close the management pane:

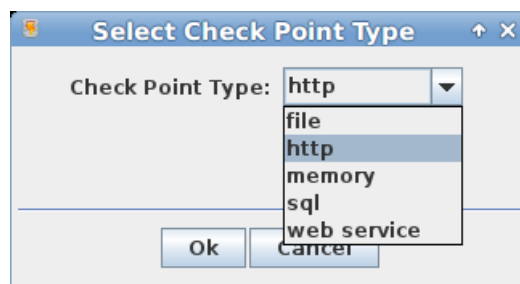


## Creating a Web Checkpoint

To create a web check point click the Checkpoint button  while on a web page of interest. In the example below the user is displaying a requisition document:



When the checkpoint button is clicked the **Select Check Point Type** dialog will display:



Web tests support multiple check point types. The check point types file, sql and web service are quite similar to the file, database and web service tests discussed previously. The memory check point provides the ability to run JVM memory tests on the JMX connection associated with the test platform.

To create an HTTP checkpoint choose **http** from the drop down and click **Ok** – this will display the **Add new checkpoint** dialog.

Checkpoint Name:

Delivery	DocumentOverview	Header Info	Items	PaymentInfo	RouteLog	Vendor
AccountSummary	AdHocRecipients	AdditionalInstitutionalInfo	CapitalAsset			
Use	Section	Property Name	Type	Operator	Value	On Failure
<input type="checkbox"/>	Current Items: Item 1 - account[1]	*Chart		equal to	UA	
<input type="checkbox"/>	Current Items: Item 1 - account[1]	*Account Number		equal to	2232800	
<input type="checkbox"/>	Current Items: Item 1 - account[1]	Sub-Account				
<input type="checkbox"/>	Current Items: Item 1 - account[1]	*Object		equal to	5490	
<input type="checkbox"/>	Current Items: Item 1 - account[1]	Sub-Object				
<input type="checkbox"/>	Current Items: Item 1 - account[1]	Project				
<input type="checkbox"/>	Current Items: Item 1	Item Line #		equal to	1	
<input type="checkbox"/>	Current Items: Item 1	Item Type		equal to	QUANTITY TAXABLE	
<input type="checkbox"/>	Current Items: Item 1	Quantity		equal to	1.00	
<input type="checkbox"/>	Current Items: Item 1	UOM		equal to	EAEACH	
<input type="checkbox"/>	Current Items: Item 1	Catalog #		equal to	216470	
<input type="checkbox"/>	Current Items: Item 1	Description		equal to	Lasko 2510 Tower Fan	
<input type="checkbox"/>	Current Items: Item 1	Unit Cost		equal to	51.43	
<input type="checkbox"/>	Current Items: Item 1	Extended Cost		equal to	51.43	
<input type="checkbox"/>	Current Items: Item 1	Tax Amount		equal to	4.68	
<input type="checkbox"/>	Current Items: Item 1	Total Amount		equal to	56.11	
<input type="checkbox"/>	Current Items: Item 1	Actions				
<input type="checkbox"/>	Current Items: Item 1	*Percent		equal to	100	
<input type="checkbox"/>	Current Items: Item 2 - account[1]	*Chart		equal to	UA	
<input type="checkbox"/>	Current Items: Item 2 - account[1]	*Account Number		equal to	2232800	
<input type="checkbox"/>	Current Items: Item 2 - account[1]	Sub-Account				
<input type="checkbox"/>	Current Items: Item 2 - account[1]	*Object		equal to	5230	
<input type="checkbox"/>	Current Items: Item 2 - account[1]	Sub-Object				
<input type="checkbox"/>	Current Items: Item 2 - account[1]	Project				
<input type="checkbox"/>	Current Items: Item 2	Item Line #		equal to	2	
<input type="checkbox"/>	Current Items: Item 2	Item Type		equal to	QUANTITY TAXABLE	
<input type="checkbox"/>	Current Items: Item 2	Quantity		equal to	1.00	
<input type="checkbox"/>	Current Items: Item 2	UOM		equal to	PKPACKET	
<input type="checkbox"/>	Current Items: Item 2	Catalog #		equal to	618405	

The layout of the HTTP check point dialog will vary depending on the source HTML page. Layout configuration (the way the HTML tags are displayed on the check point screen) is determined by a set of configurable tag handler objects that use XML tag handler definitions as input. This design allows the kuali test application to handle complex web pages in a flexible, generic manner. In the example above, there are tag handlers configured to support the KFS tab-based document display. Complete tag handler configuration is found in **Appendix A**, example XML snippets are shown below:

### Default text input tag handler

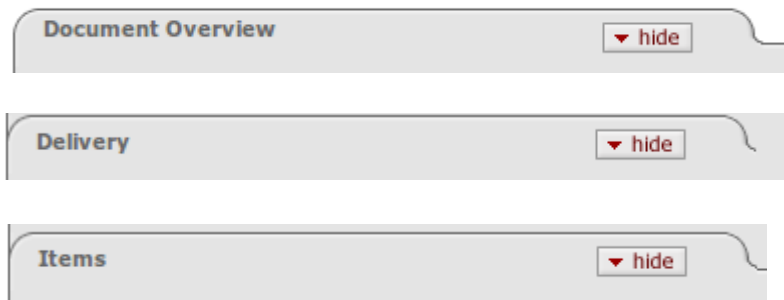
```
<handler>
  <tag-name>input</tag-name>
  <handler-class-name>
    org.kuali.test.handlers.TextInputTagHandler
  </handler-class-name>
  <tag-matchers>
    <tag-matcher>
      <match-type>current</match-type>
      <tag-name>input</tag-name>
      <match-attributes>
        <match-attribute>
          <name>type</name>
          <value>text</value>
        </match-attribute>
      </match-attributes>
    </tag-matcher>
  </tag-matchers>
</handler>
```

## Capital Asset Tab System Selection tag handler

```
<handler>
  <application>KFS</application>
  <tag-name>td</tag-name>
  <handler-class-name>
    org.kuali.test.handlers.TdTagHandler
  </handler-class-name>
  <tag-matchers>
    <tag-matcher>
      <match-type>current</match-type>
      <tag-name>td</tag-name>
      <match-attributes></match-attributes>
    </tag-matcher>
    <tag-matcher>
      <match-type>parent</match-type>
      <tag-name>table</tag-name>
      <match-attributes>
        <match-attribute>
          <name>class</name>
          <value>datatable</value>
        </match-attribute>
        <match-attribute>
          <name>summary</name>
          <value>System Selection</value>
        </match-attribute>
      </match-attributes>
    </tag-matcher>
    <tag-matcher>
      <match-type>sibling</match-type>
      <tag-name>th</tag-name>
      <search-definition>-1</search-definition>
      <match-attributes></match-attributes>
    </tag-matcher>
  </tag-matchers>
  <label-matcher>
    <tag-matcher>
      <match-type>sibling</match-type>
      <tag-name>th</tag-name>
      <search-definition>-1</search-definition>
      <match-attributes></match-attributes>
    </tag-matcher>
  </label-matcher>
  <handler-name>kfs2</handler-name>
</handler>
```

In the current example the check point dialog is displaying requisition document data. The tag handlers have been configured to display this data as follows:

1. Kuali KFS Tabs such as those shown below



will show as Tabs in the checkpoint dialog

Delivery	DocumentOverview	Header Info	Items	PaymentInfo	R
AccountSummary		AdHocRecipients			
Use	Section	Property Name			
<input type="checkbox"/>		*Description:			
<input type="checkbox"/>		Explanation:			
<input type="checkbox"/>		Org. Doc. #:			
<input type="checkbox"/>		*Year:			
<input type="checkbox"/>		Total Amount:			
<input type="checkbox"/>		*Chart/Org:			
<input type="checkbox"/>		*Funding Source:			
<input type="checkbox"/>		Receiving Required:			
<input type="checkbox"/>		Payment Request Po...			
<input type="checkbox"/>		Use Tax Indicator:			

2. Header values on the KFS document tabs such as those shown below



will show under the **Section** column in the checkpoint dialog

Delivery		Document Overview		Header Info	Items	Payment Info	Route Log	Vendor
Account Summary				Ad Hoc Recipients			Additional Institutions	
Use	Section			Property Name		Type	Operator	
<input type="checkbox"/>	Vendor Address			Suggested Vendor:			equal to	
<input type="checkbox"/>	Vendor Address			City:			equal to	
<input type="checkbox"/>	Vendor Address			Vendor #:			equal to	
<input type="checkbox"/>	Vendor Address			State:			equal to	
<input type="checkbox"/>	Vendor Address			Address 1:			equal to	
<input type="checkbox"/>	Vendor Address			Province:				
<input type="checkbox"/>	Vendor Address			Address 2:				
<input type="checkbox"/>	Vendor Address			Postal Code:			equal to	
<input type="checkbox"/>	Vendor Address			Attention:				
<input type="checkbox"/>	Vendor Address			Country:			equal to	
<input type="checkbox"/>	Vendor Info			Customer #:				
<input type="checkbox"/>	Vendor Info			Notes To Vendor:				
<input type="checkbox"/>	Vendor Info			Payment Terms:			equal to	
<input type="checkbox"/>	Vendor Info			Shipping Title:				
<input type="checkbox"/>	Vendor Info			Shipping Payment Te...				
<input type="checkbox"/>	Vendor Info			Contract Name:			equal to	
<input type="checkbox"/>	Vendor Info			Contacts:				
<input type="checkbox"/>	Vendor Info			Phone Number:				
<input type="checkbox"/>	Vendor Info			Supplier Diversity:			equal to	
<input type="checkbox"/>	Vendor Info			Fax Number:			equal to	

3. Data labels such as the ones shown below

<b>Suggested Vendor:</b>
<b>Vendor #:</b>
<b>Address 1:</b>
<b>Address 2:</b>
<b>Attention:</b>

will show under the **Property Name** column in the checkpoint dialog

Delivery	DocumentOverview	Header Info	Items	PaymentInfo	RouteLog	Vendor
AccountSummary	AdHocRecipients	AdditionalInstitu				
Use	Section	Property Name	Type	Operator		
<input type="checkbox"/>	Vendor Address	Suggested Vendor:		equal to		
<input type="checkbox"/>	Vendor Address	City:		equal to		
<input type="checkbox"/>	Vendor Address	Vendor #:		equal to		
<input type="checkbox"/>	Vendor Address	State:		equal to		
<input type="checkbox"/>	Vendor Address	Address 1:		equal to		
<input type="checkbox"/>	Vendor Address	Province:				
<input type="checkbox"/>	Vendor Address	Address 2:				
<input type="checkbox"/>	Vendor Address	Postal Code:		equal to		
<input type="checkbox"/>	Vendor Address	Attention:				
<input type="checkbox"/>	Vendor Address	Country:		equal to		
<input type="checkbox"/>	Vendor Info	Customer #:				
<input type="checkbox"/>	Vendor Info	Notes To Vendor:				
<input type="checkbox"/>	Vendor Info	Payment Terms:		equal to		
<input type="checkbox"/>	Vendor Info	Shipping Title:				
<input type="checkbox"/>	Vendor Info	Shipping Payment Te...				
<input type="checkbox"/>	Vendor Info	Contract Name:		equal to		
<input type="checkbox"/>	Vendor Info	Contacts:				
<input type="checkbox"/>	Vendor Info	Phone Number:				
<input type="checkbox"/>	Vendor Info	Supplier Diversity:		equal to		
<input type="checkbox"/>	Vendor Info	Fax Number:		equal to		

## Creating Web Checkpoint Properties

A web check point can contain multiple check point properties. The check point properties will be evaluated in future test runs to determine test pass/fail status. To select check point properties pull up the web check point dialog and check the **Use** check box for the desired properties then select valid entries for the **Type**, **Operator**, **Value** and **On Failure** columns.

Use	Section	Property Name	Type	Operator	Value	On Failure
<input checked="" type="checkbox"/>	Account Summary 1	Chart	string	equal to	UA	error - continue
<input checked="" type="checkbox"/>	Account Summary 1	Account Number	string	equal to	2232800	error - continue
<input type="checkbox"/>	Account Summary 1	Sub-Account				
<input checked="" type="checkbox"/>	Account Summary 1	Object	string	equal to	5230	error - continue
<input type="checkbox"/>	Account Summary 1	Sub-Object				
<input type="checkbox"/>	Account Summary 1	Project				
<input type="checkbox"/>	Account Summary 1	Org Ref Id				
<input type="checkbox"/>	Account Summary 1	Org. Doc. #				
<input checked="" type="checkbox"/>	Account Summary 1	Amt	double	equal to	12.94	error - continue
<input checked="" type="checkbox"/>	Account Summary 2	Chart	string	equal to	UA	error - continue
<input checked="" type="checkbox"/>	Account Summary 2	Account Number		equal to	2232800	
<input type="checkbox"/>	Account Summary 2	Sub-Account				

Enter a check point name and click the **Save** button to save the check point.

A web test can contain multiple check points of varying types. When the test is run the check points will be evaluated against the current test execution values.

## Creating Web Test Execution Parameters

During a web application test there are times when we must use a value from the current test execution context in web operations later in the test. Consider the example scenario below:

1. Pull up the KFS application and login
2. Create a new requisition and save
3. Perform other KFS web operations that would impact the new requisition
4. Pull up the requisition and create a check point to test tax amount and other account information.

In the scenario above when the original test is created the document number of the requisition would be saved in the test file. When the test is run we would not want to use the original document number but the new document number created during the current test run. A test execution parameter can be used to solve this problem.

Again consider the example outlined above. Assume we are sitting in the test application at the create requisition screen as shown below:

The screenshot displays the Kuali Test Framework interface. The main window shows the 'New Web Test' configuration for a test named 'new test' on the 'kfs-dev' platform. The test steps include 'Cancel', 'Checkpoint', 'Parameter', and 'Save'. The test is currently running, and the 'Document Overview' section is visible. The 'Document Overview' section contains fields for 'Description', 'Org. Doc. #', and 'Explanation'. The 'Financial Document Detail' section shows the 'Year' as 2015. The 'Requisition Detail' section includes fields for 'Chart/Org' (UA / 9507), 'Funding Source' (INSTITUTION ACC), 'Receiving Required' (checkbox), and 'Payment Request Positive Approval Required' (checkbox). The 'Delivery' section shows the 'Final Delivery' details, including 'Delivery Campus' (MC), 'Building' (building not found), 'Route Code', 'Address 1', 'Address 2', 'Room', 'Delivery To' (Robert B Tucker), 'Phone Number' (520-621-0269), 'Email' (rbtucker@email.arizona.edu), 'Date Required', and 'Date Required Reason'. The URL at the bottom is <http://uits.arizona.edu/departments/the247>.

As you can see the generated document number for the new requisition is shown in the upper right corner:

Logged in User: rbtucker		logout	
	<b>Doc Nbr:</b>	3068813	
	<b>Initiator:</b>	rbtucker	
	<b>Requisition #:</b>	Not Available	R

This is the number we would like to grab and pass along during the test. To do this click the **Parameter** button



To display the **Add new test execution parameter** dialog.

Add new test execution parameter

Parameter Name: accountNumber

Parameter Value:

Parameters

Remove Parameter

Parameter Name	Parameter Value

Save

Cancel

Select the appropriate **Parameter Name** from the drop down list. The parameter name setup is discussed in the **Test Creator Tool Bar and Main Menu** under the **Getting Started** section of this document.

For this scenario we would choose the parameter name documentNumber.



**Add new test execution parameter**

Parameter Name:

Parameter Value:

**Parameters**

Parameter Name:

Value:

Now click the lookup icon on the  **Parameter Value** field to display the **Test Execution Parameter Select** dialog, choose the row with **Doc Nbr:** under the **Name** column and click **Ok** to close the dialog.

**Test Execution Parameter Select**

**Document Overview** **Header Info** **Items** **Payment Info** **Route Log** **Vendor**

**AdHocRecipients** **AdditionalInstitutionalInfo** **CapitalAsset** **Delivery**

Section	Name	Current Value
	Created:	02:47 PM 07/07/2014
	<b>Doc Nbr:</b>	<b>3068835</b>
	Initiator:	rbtucker
	Requisition #:	Not Available
	Requisition Status:	In Process
	Status:	INITIATED

**Add new test execution parameter**

Parameter Name:

Parameter Value:

**Parameters**

Remove Parameter

Parameter Name	Parameter Value

Any existing parameters will display in the **Parameters** table. Once set, all HTTP requests containing the parameter documentNumber will have the value set to the number associated with the screen value as described in the scenario above. To clear a test execution parameter, click the **Parameter** button



select the parameter you wish to remove from the **Parameters** table, click the **Remove Parameter** button then click **Save**.

In cases where a parameter name is not standardized and may have several names, for example – documentNumber, docId, docNbr – just repeat the process described above for each name.

## Test Results Delivery

Test results are delivered via email and can be configured at several levels.

### **Global**

From the main menu go to **File->Setup->Email Setup** to display the **Application Email Setup** dialog. Enter a comma delimited list of email addresses that will receive test results from all tests. This entry can be left empty.

### **Platform**

On the platform **Add/Edit Platform** dialog enter a comma delimited list of email addresses that will receive test results form all tests associated with this platform. This field can be left empty.

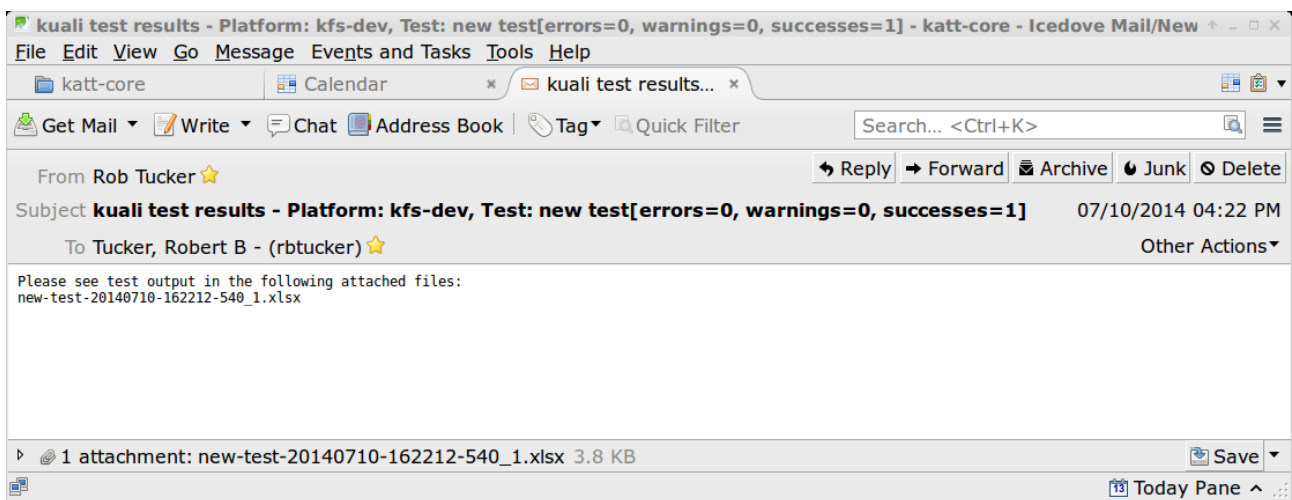
## Test Suite

On the **Add/Edit Test Suite** dialog enter a comma delimited list of email addresses that will receive test results associated with this test suite. This file can be left empty.

## Example Test Result

A test result email will contain an Excel spreadsheet attachment with test result details and may contain SQL results in CSV format for database tests and SQL checkpoints if specified.

Below is an example email and Excel attachment:



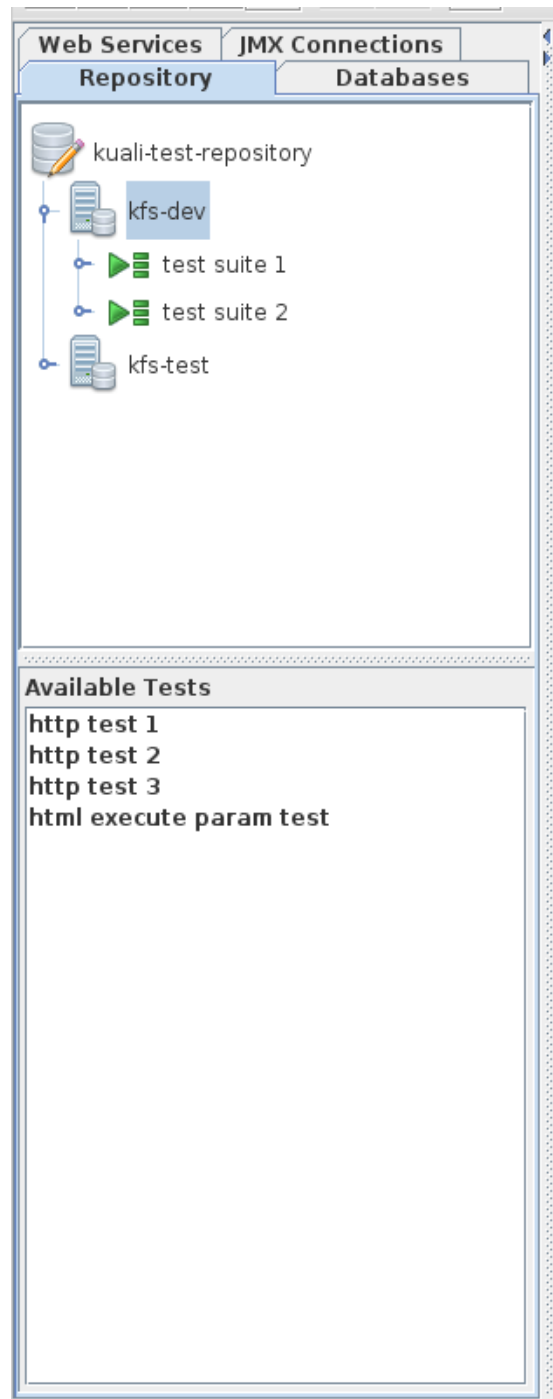
The screenshot shows an Excel spreadsheet titled "new-test-20140710-162212-540\_1.xlsx - Gnumeric". The spreadsheet contains the following data:

Checkpoint Name	Checkpoint Type	Group	Section	Start Time	End Time	Run Time (sec)	Expected Values	Actual Values	Status	Errors
new checkpoint	sql	database		16:22:13	16:22:13	0	Row Count > 0	Row Count = 763	success	

The spreadsheet also includes a summary row at the bottom: "kualitest" with a "Sum=0" value.

## Test Management Panel

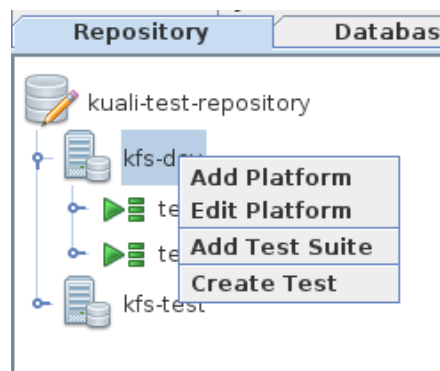
The test management panel provides the user interface to manage the various test entities created by the application and consists of 4 tabs – **Repository**, **Databases**, **Web Services** and **JMX Connections**:



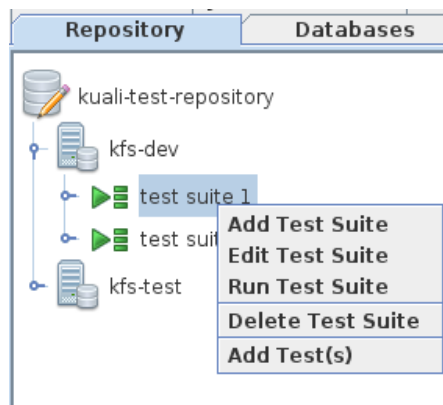
## Repository

The repository tab is a logical representation of the test repository. The upper pane contains a tree view of the test repository. The lower pane lists available platform tests for a selected platform. Most functionality is driven by a popup menu. Several of which are shown below. When a test is created for a platform it will appear in the **Available Tests** list. Test can be run individually or added to a Test Suite. To add available tests to a test suite, create the test suite and drag the tests from the available tests list to the desired test suite in the repository tree.

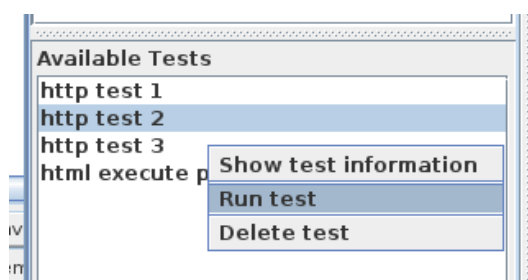
### Platform Popup Menu



### Test Suite Popup Menu

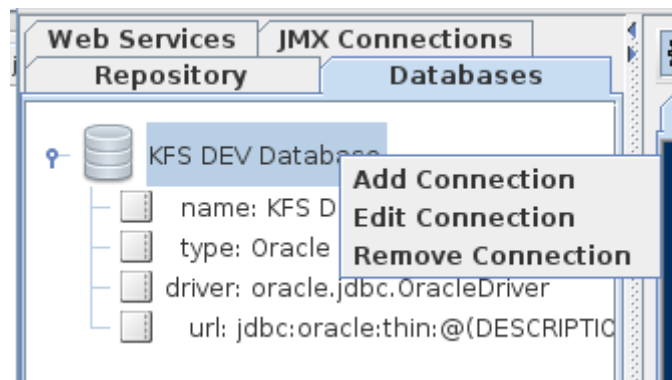


### Test Popup Menu



## Databases

The databases tab displays a tree view of all JDBC connections setup in the application. Functionality is driven by a popup menu.



## Web Services

The web services tab displays a tree view of all web service URLs setup in the application. Functionality is driven by a popup menu.



## JMX Connections

The JMX connections tab displays a tree view of all JMX URLs setup in the application. Functionality is driven by a popup menu.

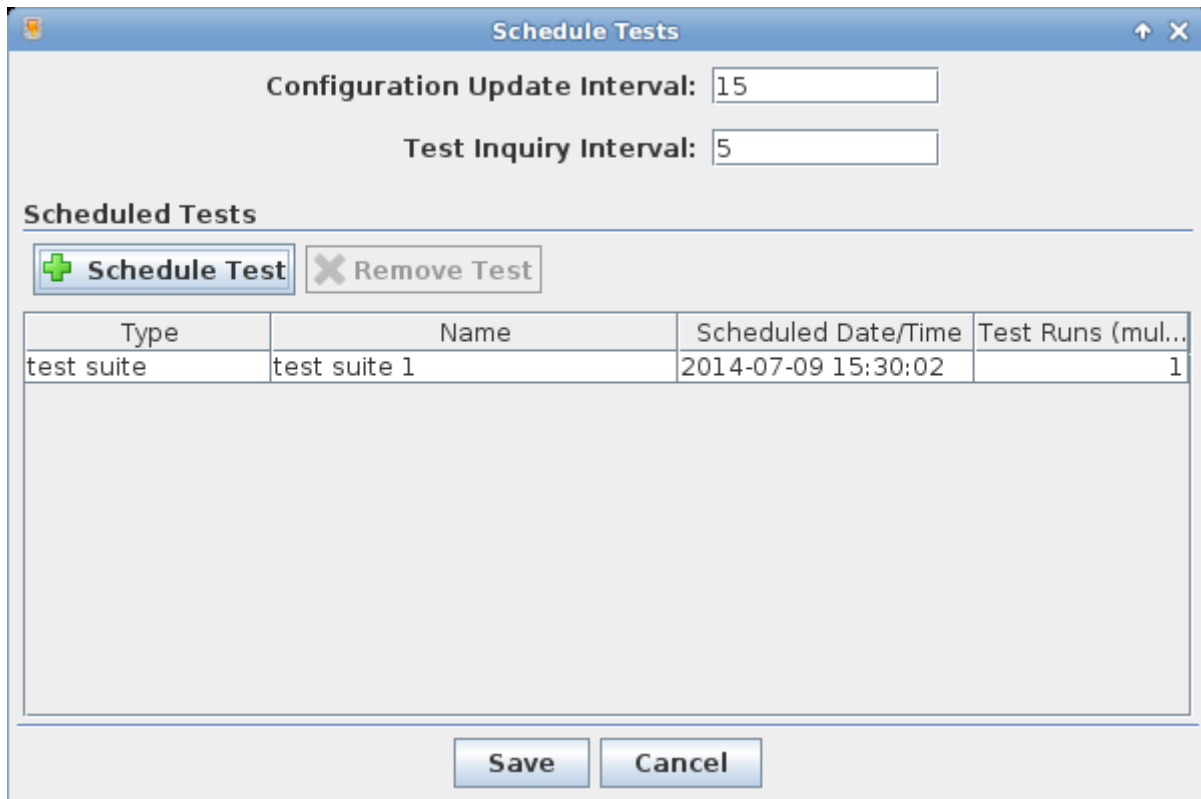


## Test Scheduling

To schedule the running of tests and test suites click **File->Schedule Tests...** menu item on the application main menu or the schedule tests tool bar button



to display the **Schedule Tests** dialog.

A screenshot of the 'Schedule Tests' dialog box. It has a title bar with a maximize button and a close button. The dialog contains two input fields: 'Configuration Update Interval' with the value '15' and 'Test Inquiry Interval' with the value '5'. Below these is a section titled 'Scheduled Tests' which contains two buttons: '+ Schedule Test' and 'X Remove Test'. Under these buttons is a table with four columns: 'Type', 'Name', 'Scheduled Date/Time', and 'Test Runs (mul...'. The table has one row with the following data: 'test suite', 'test suite 1', '2014-07-09 15:30:02', and '1'. At the bottom of the dialog are 'Save' and 'Cancel' buttons.

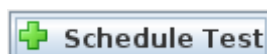
Type	Name	Scheduled Date/Time	Test Runs (mul...
test suite	test suite 1	2014-07-09 15:30:02	1

Currently scheduled tests will display in the **Scheduled Tests** table. To remove a currently scheduled test select the desired row in the table and click the **Remove Test** button.

The **Configuration Update Interval** entry defines how often in minutes that the test runner will update the schedule from the schedule file. When the schedule is modified and saved it is written as an XML file. The test runner will pick up this file at the defined interval.

The **Test Inquiry Interval** defines how often in minutes that the test runner will look for tests that are ready to run. At the defined interval any scheduled tests with a run time less than or equal to the current time will run. Once a test is run it is removed from the schedule.

To schedule a test or test suite click the **Schedule Test** button



to display the **Schedule Test** dialog.

**Schedule Test**

Platform: **kfs-dev**

Start Date/Time:

Test Runs:

**Test Suites**

- test suite 1
- test suite 2

**Platform Tests**

- http test 1
- http test 2
- http test 3
- html execute param test

**Save** **Cancel**

Choose the desired platform from the **Platform** drop down and select a Start Date/Time by clicking on the calendar icon.

**Schedule Test**

Platform: **kfs-dev**

Start Date/Time: **07/15/2014 15:44**

July 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

**Test Suites**

- test suite 1
- test suite 2

**Platform Tests**

- http test 1
- http test 2
- http test 3
- html execute param test

**Save** **Cancel**

The start date and time must be in the future. The **Test Runs** entry defaults to 1. You can enter a number greater than 1 and the test runner will spawn off multiple threads to asynchronously run the entered number of instances of the selected test/test suite.

Choose the desired platform test or test suite to run and click **Save** to schedule the test.



Schedule Test

Platform: kfs-dev

Start Date/Time: 07/15/2014 15:44

Test Runs: 20

Test Suites

test suite 1  
test suite 2

Platform Tests

http test 1  
http test 2  
http test 3  
html execute param test

Save

Cancel

Schedule Tests

Configuration Update Interval: 15

Test Inquiry Interval: 5

Scheduled Tests

+

Schedule Test

×

Remove Test

Type	Name	Scheduled Date/Time	Test Runs (mul...
test suite	test suite 1	2014-07-09 15:42:23	1
platform test	html execute param test	2014-07-15 15:44:48	20

Save

Cancel

## Appendix A: Tag Handler Configuration

To support complex HTML presentation pages in generic, flexible and extensible fashion the test application implements a set of configurable tag handlers. A tag handler is an XML configurable Java object the allows a user to specify a set of rules for uniquely identifying a an HTML tag on a web page and specifying an output presentation in the test application. This identification/presentation logic allows the test application to present a user-friendly

interface for test parameter selections during a web test.

## ***Tag Handler Java Classes***

The test application provides the following tag handlers out of the box:

### **Interface**

HtmlTagHandler

### **Abstract Base Class**

DefaultHtmlTagHandler

### **General Purpose Handlers**

CheckboxInputTagHandler  
DefaultContainerTagHandler  
NoopTagHandler  
RadioInputTagHandler  
SelectInputTagHandler  
TdTagHandler  
TextareaTagHandler  
TextInputTagHandler

### **Kuali Specific Handlers**

KualiCapitalAssetTagHandler  
KualiHeaderBoxTagHandler  
KualiHeaderInfoCellTagHandler  
KualiItemAccountTagHandler  
KualiRouteLogIframeTagHandler  
KualiTabTagHandler  
KualiWorkareaTagHandler

## ***Tag Handler XML Configuration***

A tag handle is configured via an XML entry and can support multiple configuration to target different HTML presentations. Tag handle configuration files should be placed in the location specified in the **Kuali Test Framework Configuration XML** in the tag shown below:

```
<tag-handlers-location>  
    my-html-tag-handler-definition-file-dir  
</tag-handlers-location>
```

### **Tag handler configuration file naming convention**

During a web-based test the test application parses the incoming HTML response and selects a tag handle to handle incoming tags. Multiple handlers could potentially match the same tag so the test application assigns the first matched handler to an

incoming tag. To ensure that the test application chooses the correct tag handler we must make sure that the first handler we find is the match we want. There are several methods built into the application to accomplish this. The first is the configuration file naming convention. Below are the default set of configuration files that come with the application:

custom-handlers.xml  
general-handlers.xml  
kfs-handlers.xml  
kuali-handlers.xml

When searching for a tag handler match the test application will search the configuration files in the following order:

1. application-specific handler file – the file naming convention for application-specific handler files is: <application-name>-handler.xml where <application-name> is the lowercase name of the application as displayed in the platform setup.
2. custom handlers – handler placed in the custom-handlers.xml
3. kuali-specific handlers – handlers found in the kuali-handlers.xml
4. general handler – handlers found in the general-handlers.xml

### Tag Handler XML configuration example

Below is an example of a tag handler specification:

```
<handler>
  <!--
    Populate the tag below if this handler should be associated
    only with a defined application. This tag is not required.
  -->
  <application>KFS</application>
  <!--
    HTML tag name that this handler will handle
  -->
  <tag-name>td</tag-name>
  <!--
    Java handler class
  -->
  <handler-class-name>
    org.kuali.test.handlers.KualiCapitalAssetTagHandler</handler-class-name>
  <!--
    Multiple tag handlers might match an HTML tag. The first match wins. The
    previous section describes how the match order is determined by handle file
    naming convention. An additional method of enforcing handler order is to
    assign a sort index. The sort index tag is not required. If not specified, sort
    index defaults to 1000
  -->
  <sort-index>1</sort-index>
```

<tag-matchers>

<!--

A tag matcher identifies tag match attributes which determine which HTML tags this handler will match. A handler can specify one or more tag tag-matcher elements. All tag-matcher elements must be satisfied for an HTML tag to be assigned this handler

-->

<tag-matcher>

<!--

match type specifies the HTML position of this matcher in relation to the current node. The first tag-matcher will always be the current node under evaluation. Other valid entries for the match-type element are parent, child and sibling

-->

<match-type>current</match-type>

<!--

The HTML tag this matcher will match. The first tag-matcher will always match the current node and the tag-name will be the same as the handler tag-name. Other tag-matcher elements could specify any tag-name

-->

<tag-name>td</tag-name>

<!--

The match-attributes tag specifies one or more HTML tag attributes to match. Multiple attribute names can be specified by separating with a pipe symbol, for example <name>class|style|id</name>. The value tag will accept an asterisk as a wild card, for example - <value>my\*value</value>.

-->

<match-attributes>

<match-attribute>

<name>class</name>

<value>info|line</value>

</match-attribute>

</match-attributes>

</tag-matcher>

<tag-matcher>

<match-type>parent</match-type>

<tag-name>div</tag-name>

<!--

The search-definition tag specifies how to search for non-current tag matches. In this case an asterisk move up the DOM tree looking for a parent match until you find the match or hit the root node. If the tag was specified as <search-definition>1</search-definition> the search would stop after the initial parent node. Child node tag-matcher search definitions function the same as the parent described above. Sibling search are a bit more complex and are below:

“\_\*” - compare against all previous siblings

“+\*” - compare against all following siblings

“-2” - compare against 2nd previous sibling  
“+2” - compare against 2nd following sibling  
“1” - compare against first child of parent  
“3” - compare against 3rd child of parent

There is one special search-definition used only for the label-matcher described below.

“l” - match a value by column index – used for table data labels

-->

```
<search-definition>*</search-definition>
<match-attributes>
  <match-attribute>
    <name>class</name>
    <value>tab-container</value>
  </match-attribute>
</match-attributes>
</tag-matcher>
<tag-matcher>
  <match-type>parent</match-type>
  <tag-name>table</tag-name>
  <search-definition>*</search-definition>
  <match-attributes>
    <match-attribute>
      <name>class</name>
      <value>datatable</value>
    </match-attribute>
    <match-attribute>
      <name>summary</name>
      <value>Capital Asset Items</value>
    </match-attribute>
  </match-attributes>
</tag-matcher>
</tag-matchers>
<!--
```

The label-matcher specifies a set of tag-matcher elements to specify the value that will display in the “Property Name” column in the Checkpoint Properties dialog. The tag-matcher logic of a label-matcher specifies a sequential set of instructions to navigate from the current node to the label node.

-->

```
<label-matcher>
  <tag-matcher>
    <match-type>parent</match-type>
    <tag-name>table</tag-name>
    <search-definition>1</search-definition>
    <match-attributes>
      <match-attribute>
```

```

        <name>class</name>
        <value>datatable</value>
    </match-attribute>
    <match-attribute>
        <name>summary</name>
        <value>Capital Asset Items</value>
    </match-attribute>
</match-attributes>
</tag-matcher>
<tag-matcher>
    <match-type>child</match-type>
    <tag-name>tr</tag-name>
    <search-definition>2</search-definition>
    <match-attributes></match-attributes>
    <child-tag-match>
        <child-tag-name>th</child-tag-name>
        <match-attributes></match-attributes>
    </child-tag-match>
</tag-matcher>
<tag-matcher>
    <match-type>child</match-type>
    <tag-name>th</tag-name>
    <search-definition>1</search-definition>
    <match-attributes></match-attributes>
</tag-matcher>
</label-matcher>
<!--
    The section-matcher specifies a set of tag-matcher elements to specify the
    value that will display in the "Section" column in the Checkpoint
    Properties dialog. The tag-matcher logic of a section-matcher specifies a
    sequential set of instructions to navigate from the current node to the
    section node
-->
<section-matcher>
    <tag-matcher>
        <match-type>sibling</match-type>
        <tag-name>td</tag-name>
        <search-definition>1</search-definition>
        <match-attributes></match-attributes>
    </tag-matcher>
</section-matcher>
<!--
    The handler-name tag is not required and is used mostly for debug purposes.
    You can use this to identify a tag handler that was selected for an HTML tag.
-->
    <handler-name>kfs3</handler-name>
</handler>

```

## Appendix B: Additional Database Information

The test application provides the ability to configure user-friendly table and column names, pseudo foreign key relationships and SQL where value lookup definitions for the Database tests and SQL checkpoints. These custom configurations are found in the database additional information files found in the **Kuali Test Framework Configuration XML** in tag the tag below:

```
<additional-db-info-location>
    location-of-db-additional-info-files
</additional-db-info-location>
```

The naming convention for these files is “application-name”-additional-db-info.xml where “application-name” is the lower case version of the application names defined for the platforms.

### ***Additional Database Information File Example***

```
<additional-database-info xmlns="test.kuali.org">
  <application>
    <application-name>KFS</application-name>
    <tables>
      <table>
        <!-- database table name -->
        <table-name>FP_DV_EXT_T</table-name>
        <!-- user-friendly table name -->
        <display-name>
          DisbursementVoucherDocumentExtension
        </display-name>
        <columns>
          <column>
            <column-name>
              FIN_COA_CD
            </column-name>
            <display-name>
              chartOfAccountsCode
            </display-name>
            <!--
              SQL where value lookups and be
              defined at the column level
            -->
            <lookup-sql-select>
              select fin_coa_cd, fin_coa_cd
              from ca_chart_t order by 1
            </lookup-sql-select>
          </column>
          <column>
            <!-- database column name -->
```

```

        <column-name>
            FDOC_NBR
        </column-name>
        <!-- user-friendly column name -->
        <display-name>
            documentNumber
        </display-name>
    </column>
</columns>
<!--
    define pseudo foreign key relationships – these
    would be known table relationships that do not
    have foreign key constraints in the database
-->
<custom-foreign-keys>
    <custom-foreign-key>
        <name>customfk1</name>
        <primary-table-name>
            KREW_DOC_HDR_T
        </primary-table-name>
        <foreign-key-column-pair>
            <primary-column>
                DOC_HDR_ID
            </primary-column>
            <primary-column-type>
                number
            </primary-column-type>
            <foreign-column>
                FDOC_NBR
            </foreign-column>
            <foreign-column-type>
                string
            </foreign-column-type>
        </foreign-key-column-pair>
    </custom-foreign-key>
</custom-foreign-keys>
</table>
<tables>
<!--
    global lookup definitions for SQL where column values can be
    defined based on column name
-->
<lookups>
    <lookup>
        <column-name>FIN_COA_CD</column-name>
        <sql>
            select fin_coa_cd, fin_coa_cd
            from ca_chart_t order by 1
        </sql>
    </lookup>
</lookups>

```



```
        </lookup>
    </lookups>
</application>
</additional-database-info>
```

## Appendix C: For Developers

The Kuali Test Application is a maven application that uses a number of open source libraries – a partial list is shown below:

### ***Open Source Technologies Used***

- XMLBeans - technology for accessing XML by binding it to Java types  
<http://xmlbeans.apache.org/>
- DJ Native Swing – native browser integration into Swing application  
<http://djproject.sourceforge.net>
- LittleProxy - high-performance HTTP proxy written in Java  
<http://www.littleshoot.org/littleproxy/>
- Apache POI - Java library for reading and Excel  
<http://poi.apache.org/>
- Apache Axis - Web Services / SOAP / WSDL engine  
<http://axis.apache.org/>
- JTidy – DOM parser and tool for cleaning up malformed and faulty HTML  
<http://jtidy.sourceforge.net/>
- Jasypt - java library which allows the developer to add basic encryption capabilities to his/her projects  
<http://www.jasypt.org/>
- Apache HttpComponents - A library for client-side HTTP communication built on top of HttpClient  
<http://hc.apache.org/>

### ***Libraries that Require Manual Addition to Maven Repository***

Most of the libraries required for the test application build will be downloaded automatically. The libraries for the following will need to be added to the repository manually.

- Oracle JDBC driver
- DJ Native Swing libraries

### ***Native Browser Libraries***

The embedded native browser interface requires operating system specific library files.

The maven build handles partially handles this by providing 3 build profiles – win32, win64 and linux64 – others can be added if required. The pom.xml entries below handle this:

```
<profile>
  <id>linux64</id>
  <dependencies>
    <dependency>
      <groupId>
        org.eclipse.swt.org.eclipse.swt.gtk.linux.x86_64.4.3.swt
      </groupId>
      <artifactId>org.eclipse.swt.gtk.linux.x86_64</artifactId>
      <version>${eclipse.swt.version}</version>
    </dependency>
  </dependencies>
</profile>
<profile>
  <id>win64</id>
  <dependencies>
    <dependency>
      <groupId>
        org.eclipse.swt.org.eclipse.swt.win32.win32.x86_64.4.3.swt
      </groupId>
      <artifactId>org.eclipse.swt.win32.win32.x86_64</artifactId>
      <version>${eclipse.swt.version}</version>
    </dependency>
  </dependencies>
</profile>
<profile>
  <id>win32</id>
  <dependencies>
    <dependency>
      <groupId>
        org.eclipse.swt.org.eclipse.swt.win32.win32.x86.4.3.swt
      </groupId>
      <artifactId>org.eclipse.swt.win32.win32.x86</artifactId>
      <version>${eclipse.swt.version}</version>
    </dependency>
  </dependencies>
</profile>
```

The native libraries are found in the selected org.eclipse.swt jar file and should be removed and added to the library path specified in the TestCreator startup command line.