

# Kuali Test Framework

Rob Tucker – University of Arizona

07/05/2014

## Table of Contents

Overview.....	2
Configuration and Storage.....	2
Security.....	3
Application Architecture.....	3
Logical Test Model.....	3
Getting Started.....	4
Kuali Test Framework Configuration XML.....	4
Test Creator Command Line.....	5
Test Runner Command Line.....	6
Test Creator User Interface.....	6
Test Creator Tool Bar and Main Menu.....	6
Initial Setup.....	8
Application Email.....	8
JDBC Database Connection.....	9
JMX Connection Setup.....	9
Web Service Connection Setup.....	10
Platform Setup.....	10
Additional Database Information.....	11
Creating a New Test.....	12
Creating a File Test.....	14
Completing Checkpoint Dialog.....	15
Creating a Web Service Test.....	15
Creating a Database Test.....	18
Selecting Columns for Database Query.....	19
Database associated with test platform.....	19
Database Table.....	20
Table Column.....	20
Primary Key Table Column.....	20
Building SQL Select and Order By Clause.....	21
Building SQL Where Clause.....	23
Displaying Generated SQL and Verifying Validity.....	24
Creating SQL Checkpoint.....	25
Creating a Web Application Test.....	27
Web Application Test Framework Overview.....	27
Test Creator.....	27
Test Runner.....	28
Starting a Web Application Test.....	28
Creating a Web Checkpoint.....	30
Default text input tag handler.....	32
Capital Asset Tab System Selection tag handler.....	33
Creating Web Checkpoint Properties.....	35
Creating Web Test Execution Parameters.....	36
Test Management Panel.....	39
Repository.....	40

Platform Popup Menu.....	41
Test Suite Popup Menu.....	41
Test Popup Menu.....	41
Databases.....	41
Web Services.....	42
JMX Connections.....	42
Test Scheduling.....	43
Tag Handler Configuration Detail.....	45

## 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 twat framework is primarily designed to support 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 require 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 consists 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:

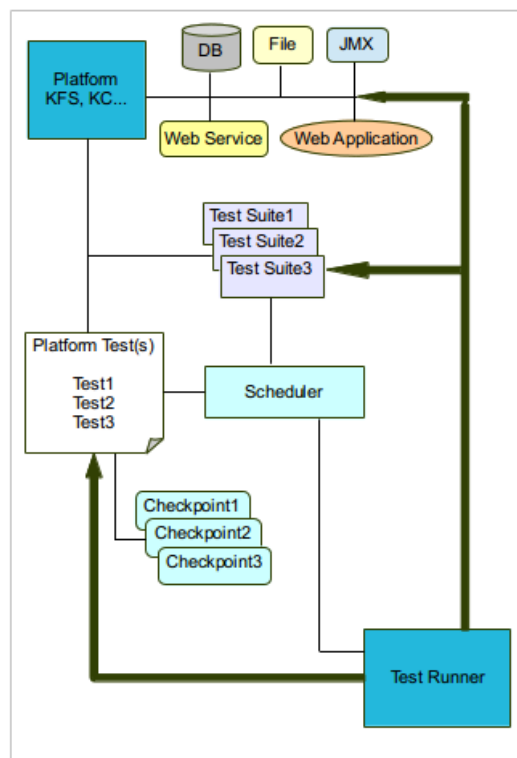


Diagram symbols are described below:

- **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.

```

javaw -Dnetwork.proxy_host=localhost -Dnetwork.proxy_port=8888
    org.kuali.test.creator.TestCreator "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-configuration-file” argument is the full path to the kuali test configuration file discussed in the previous section.

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

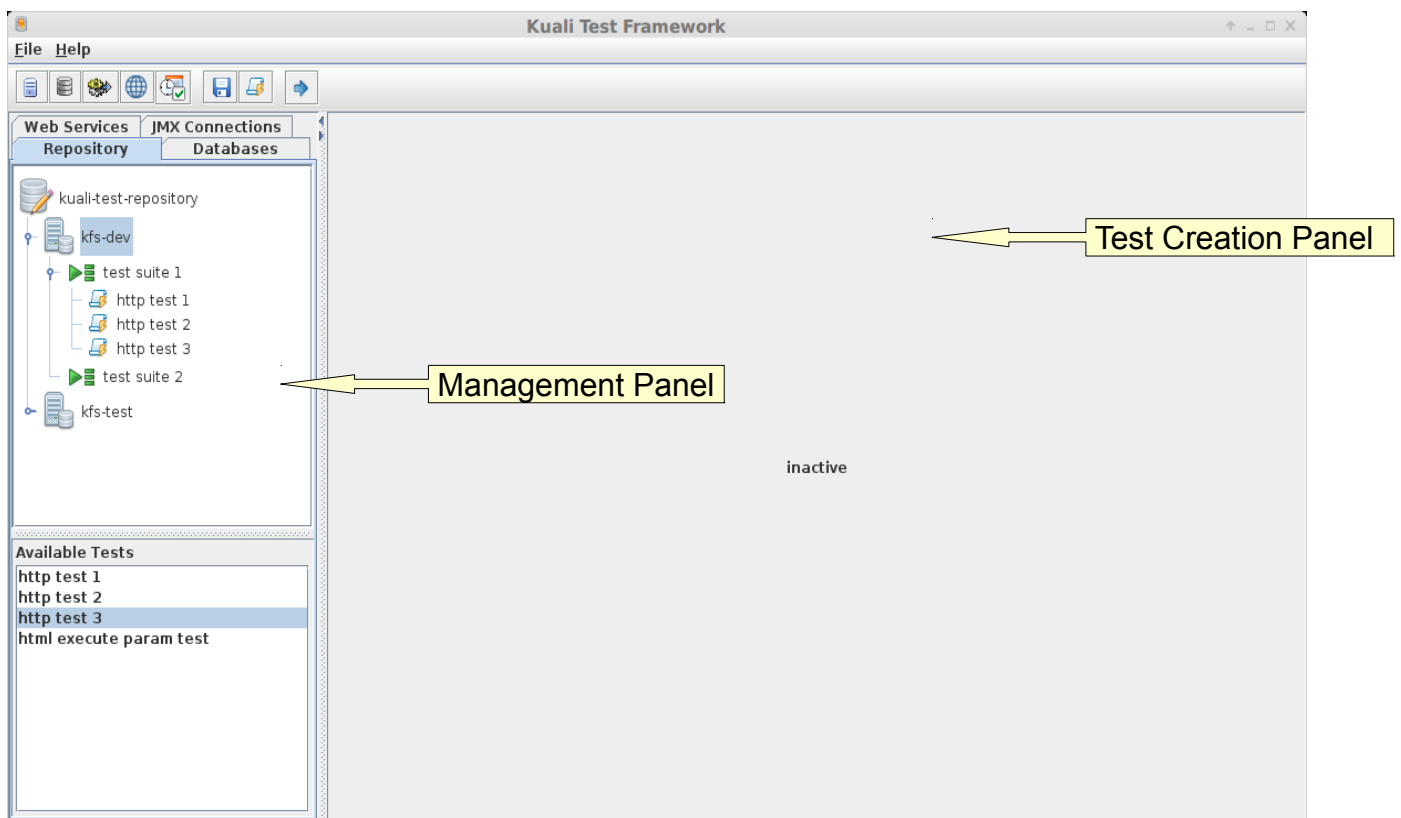
Below is an example of the command line to start and run the test runner. It is expected that the kualitest-1.0.jar can be found on the classpath.

```
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.

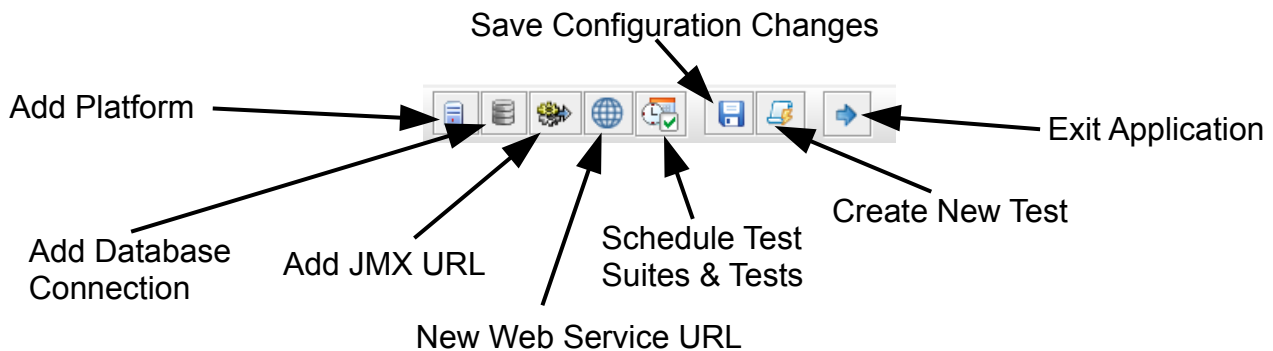
### ***Test Creator User Interface***

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.

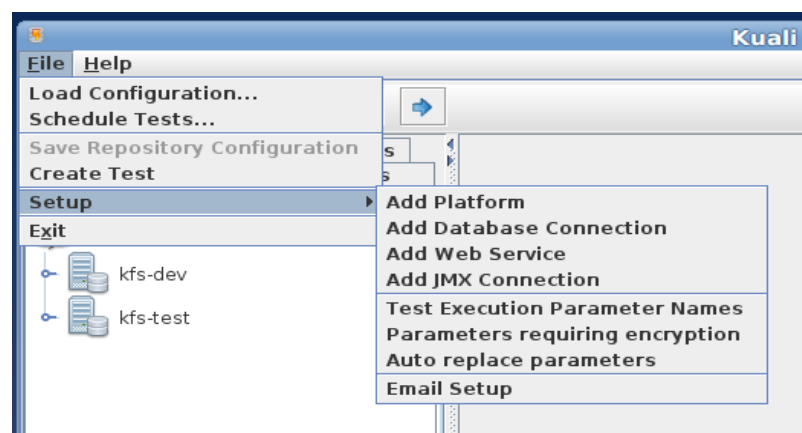


### ***Test Creator Tool Bar and Main Menu***

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=EIXikoD3RTVn36VP3CdBuZGrWyKmlTh+”
- **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.

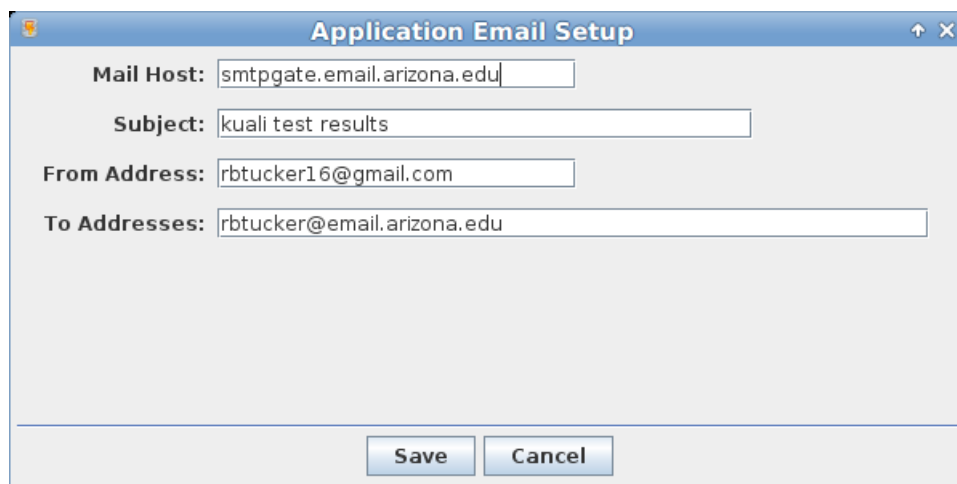
## Initial Setup

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". It contains the following fields and values:

- Mail Host:** smtpgate.email.arizona.edu
- Subject:** kuali test results
- From Address:** rbtucker16@gmail.com
- To Addresses:** rbtucker@email.arizona.edu

At the bottom of the window 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:

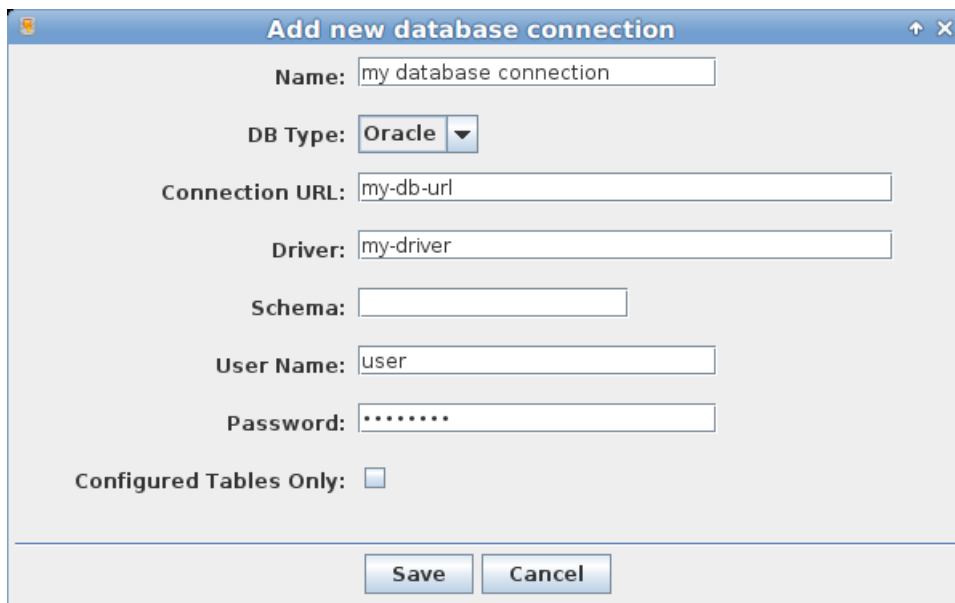
```
kuali test results - Platform: kfs-dev, Test: http test 1
```



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:



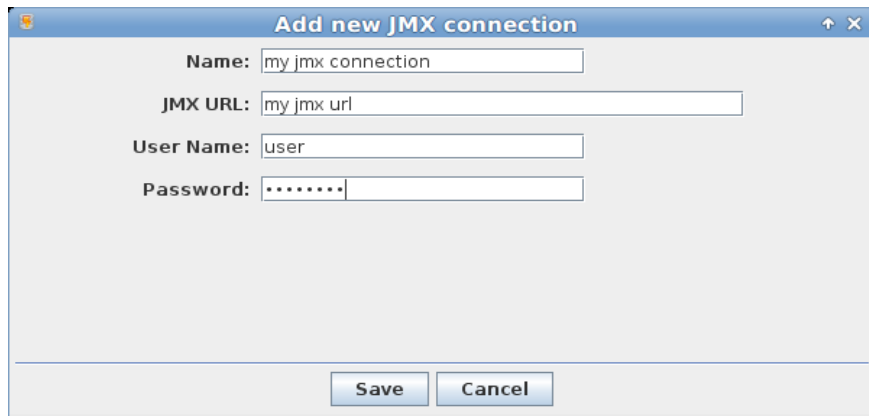
The screenshot shows a dialog box titled "Add new database connection". It contains the following fields and controls:

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

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***

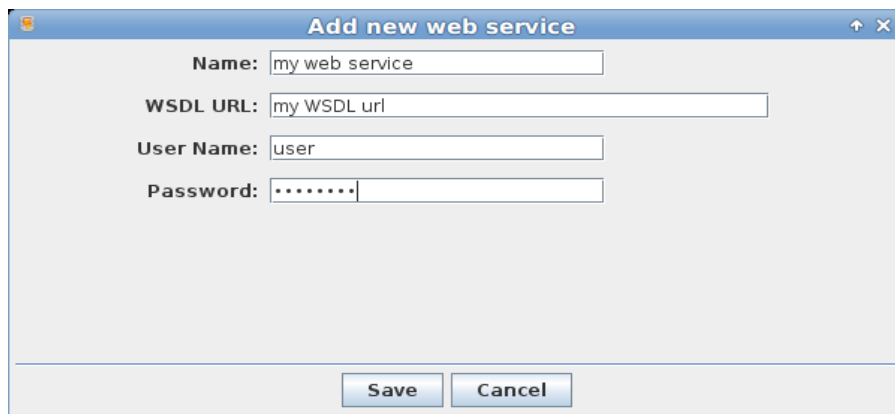
From the application main menu click on the **File->Setup->Add JMX Connection** menu item or the **Add JMX Connection** tool bar button to display the JMX connection setup dialog:



A screenshot of a dialog box titled "Add new JMX connection". It contains four input fields: "Name" with the text "my jmx connection", "JMX URL" with the text "my jmx url", "User Name" with the text "user", and "Password" with masked characters ".....". At the bottom right, 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 screenshot of a dialog box titled "Add new web service". It contains four input fields: "Name" with the text "my web service", "WSDL URL" with the text "my WSDL url", "User Name" with the text "user", and "Password" with masked characters ".....". At the bottom right, 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:

**Add new platform**

Name:

Application:

Version:

Web URL:

Web Service:

JMX URL:

Email Addresses:

DB Connection:

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>
```

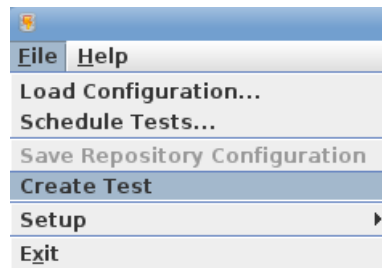
```

<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>
<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>

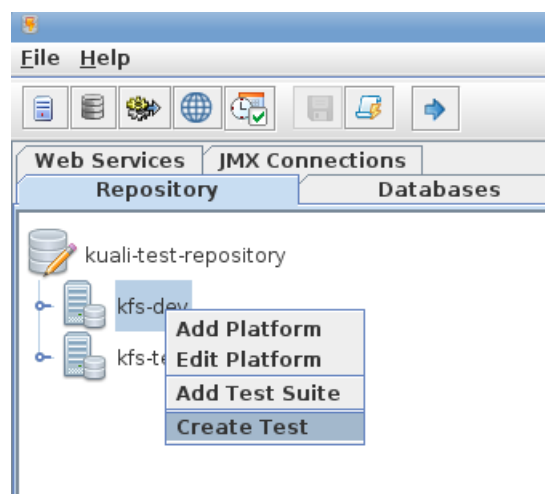
```

## 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:

A screenshot of the 'Initialize New Test' dialog box. It contains the following fields and controls:

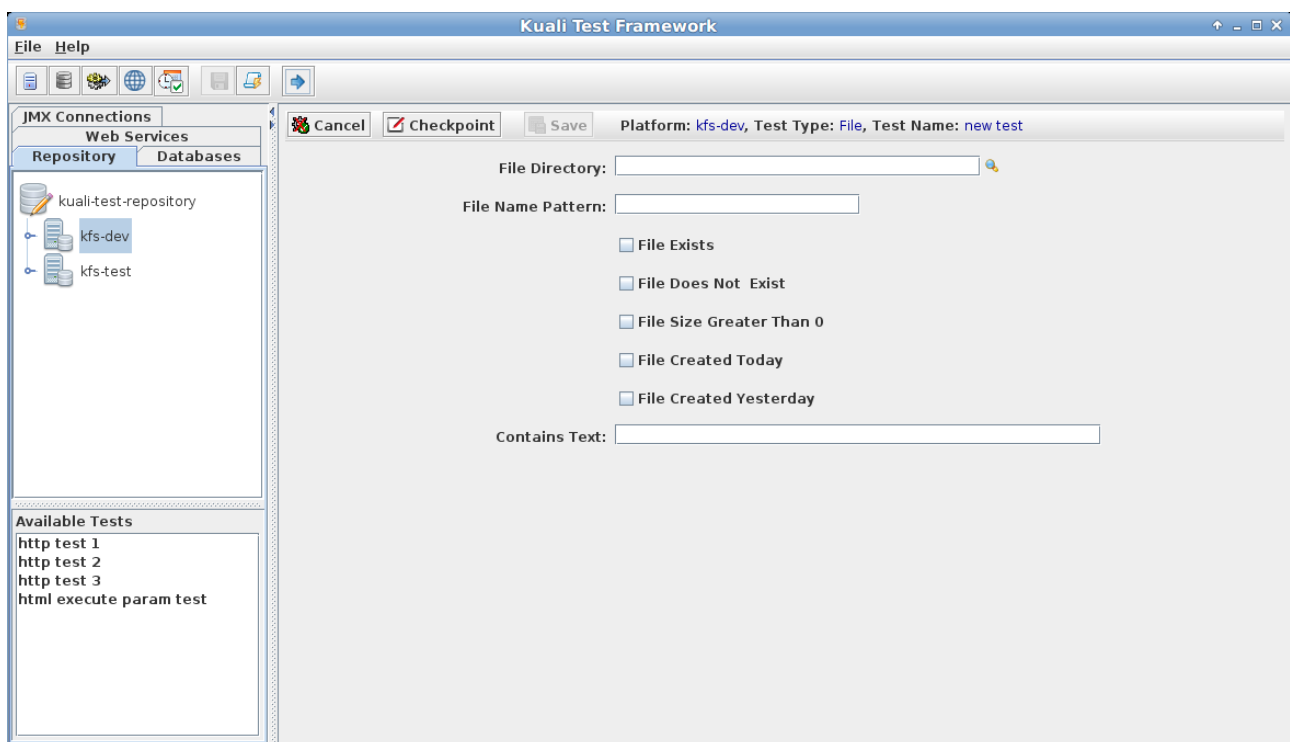
- Platform: A dropdown menu with 'kfs-dev' selected.
- Test Name: A text input field containing 'new test'.
- Test Type: A dropdown menu with 'Web' selected.
- Description: A text input field containing 'new test description'.
- Max Run Time (sec): An empty text input field.
- On Max Time Failure: A dropdown menu with a downward arrow.
- Buttons: 'Continue' and 'Cancel' buttons at the bottom.

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.

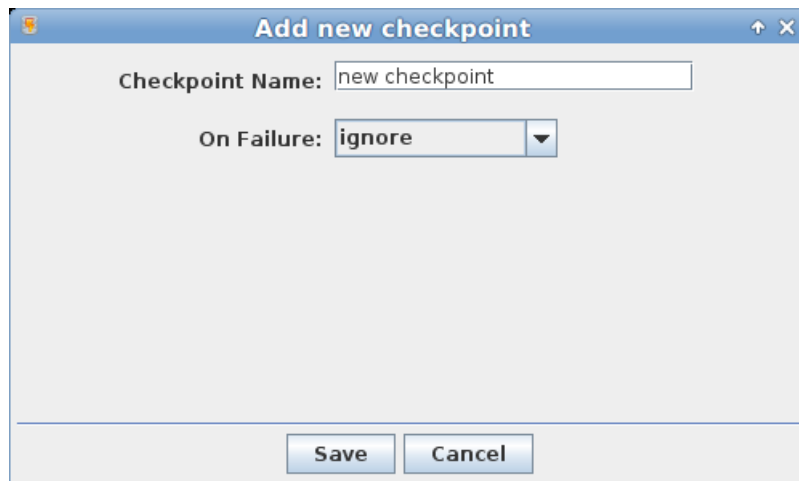
The File test panel entry fields are described below:

- **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:

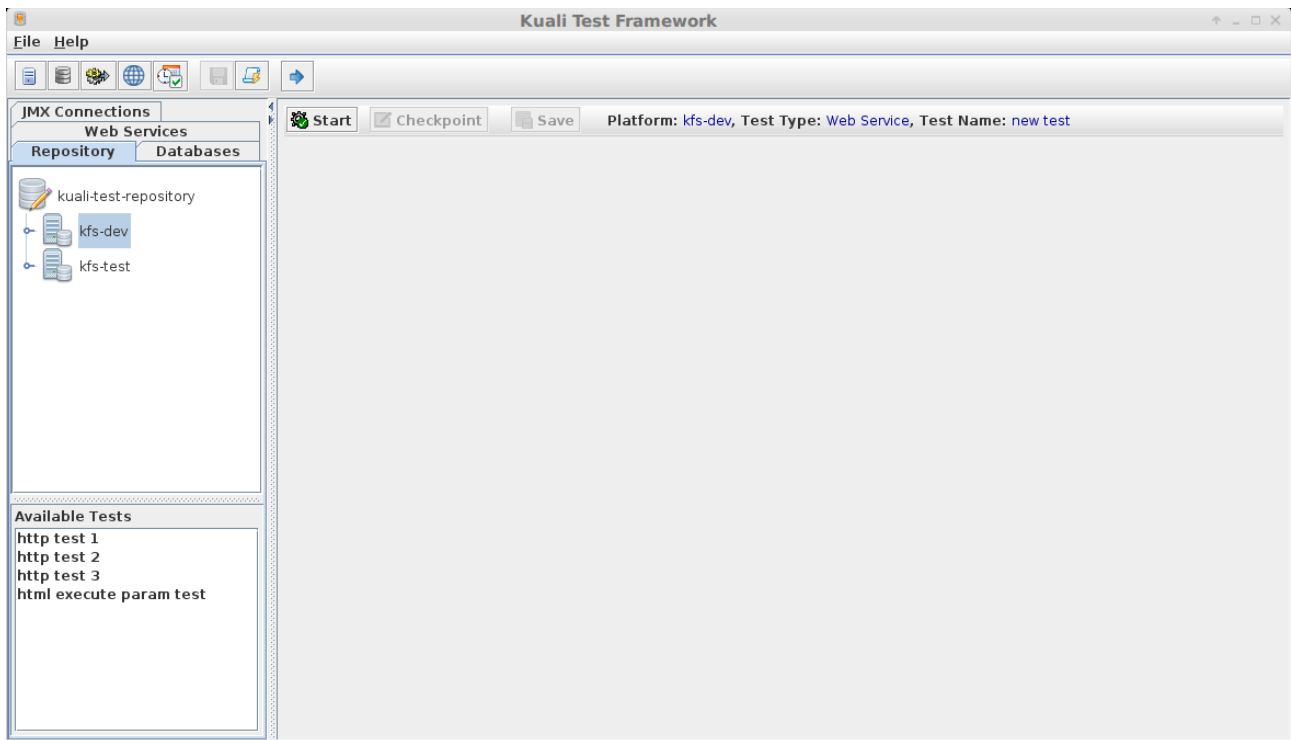


The screenshot shows a standard Windows-style dialog box titled "Add new checkpoint". It features a light gray background and a blue title bar with standard window controls. The main area contains two labels: "Checkpoint Name:" followed by a text input field containing "new checkpoint", and "On Failure:" followed by a dropdown menu currently set to "ignore". At the bottom of the dialog, there are two buttons: "Save" and "Cancel".

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

## **Creating a Web Service Test**

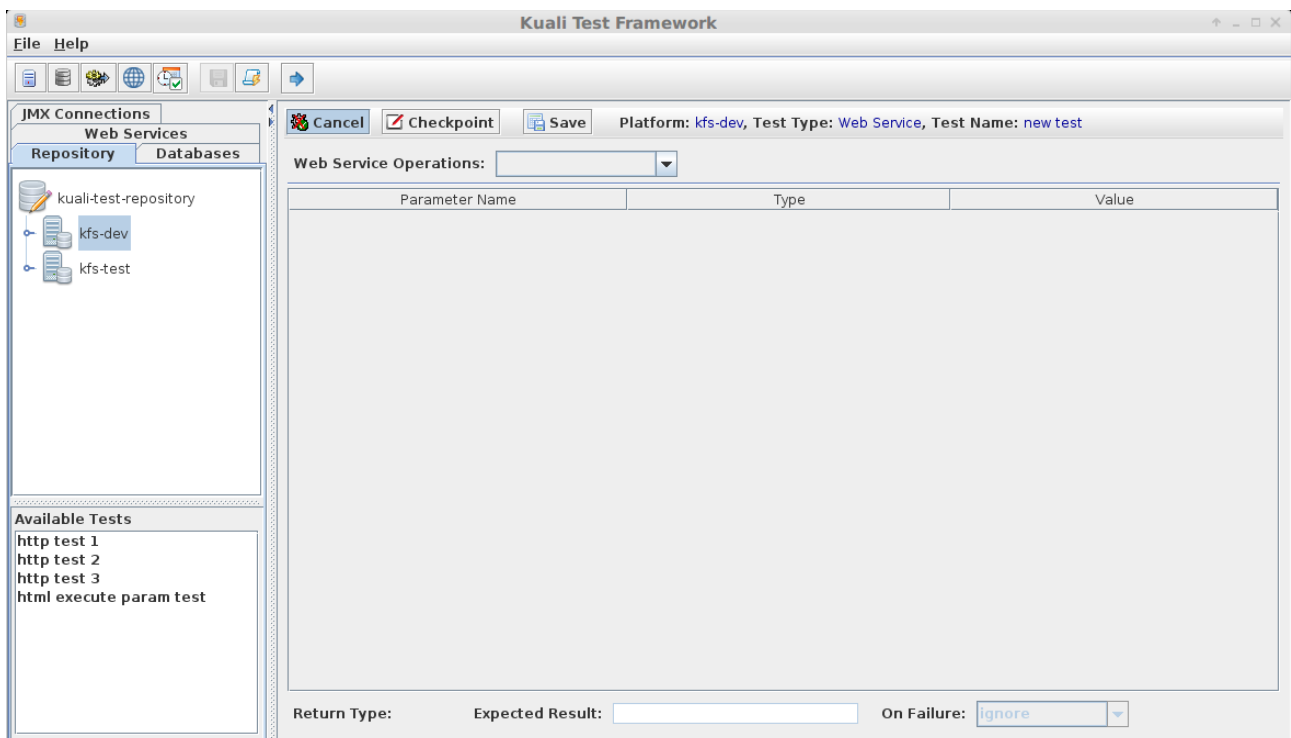
Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Web Service** is selected from the drop down. Click **Continue** to load the initial web service setup panel.



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.

The screenshot shows a software interface with a toolbar at the top containing 'Cancel', 'Checkpoint', and 'Save' buttons. To the right of the toolbar, it says 'Platform: kfs-dev, Test Type: Web Service, Test Name: new test'. Below the toolbar, there is a section labeled 'Web Service Operations:' with a dropdown menu currently showing 'GetWeather'. A context menu is open over this dropdown, listing three options: 'GetCitiesByCountry', 'GetWeather' (which is highlighted), and another 'GetWeather' option. Below the dropdown, there is a table with two columns: 'Parameter Name' and 'Type'. The table contains two rows: one for 'CityName' with type 'string' and one for 'CountryName' with type 'string'.

Complete the input parameter entries:

This screenshot shows the same software interface as before, but the 'Web Service Operations' dropdown is still set to 'GetWeather'. Below it, a table is displayed with three columns: 'Parameter Name', 'Type', and 'Value'. The table has two data rows: the first row has 'CityName' as the parameter name, 'string' as the type, and 'Tucson' as the value; the second row has 'CountryName' as the parameter name, 'string' as the type, and 'USA' as the value.

And the expected result field:

The screenshot shows a section of the software interface with three fields. The first field is labeled 'Return Type:' and contains the text 'string'. The second field is labeled 'Expected Result:' and contains the text 'sunny'. The third field is labeled 'On Failure:' and contains a dropdown menu with 'warning' selected.

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 standard Windows window frame. 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 of the dialog, there are two buttons: 'Save' and 'Cancel'.

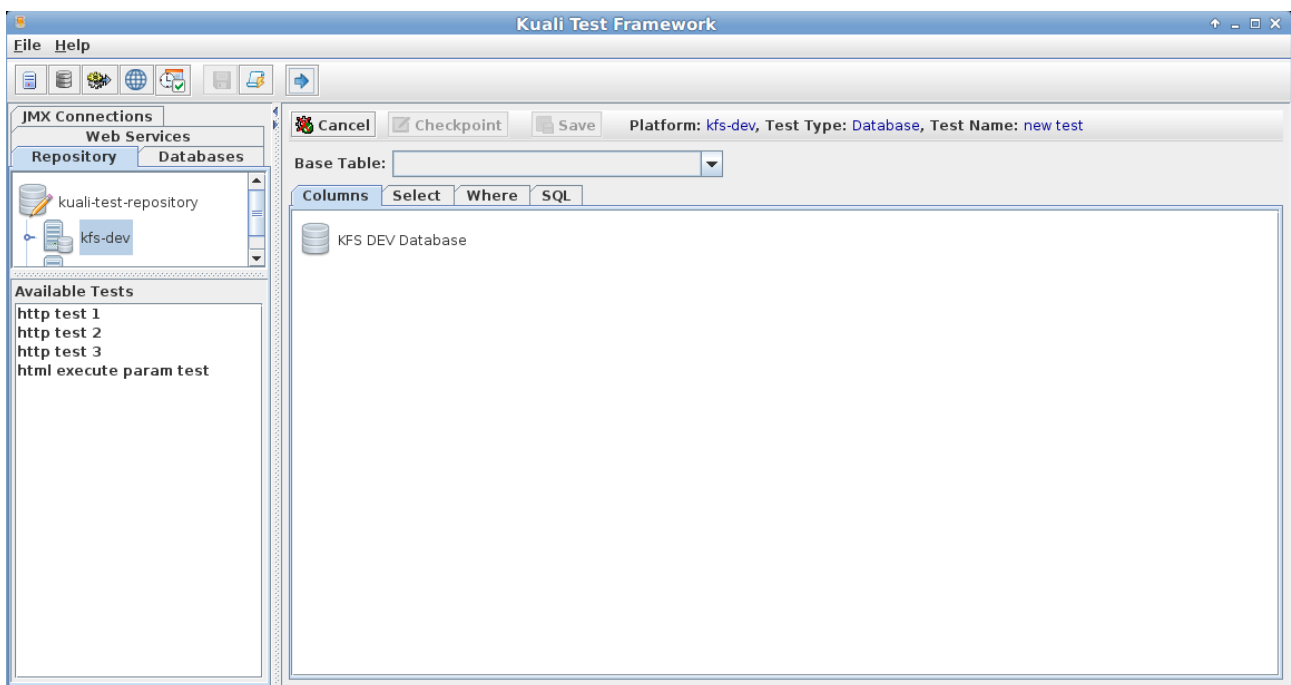
Enter **Checkpoint Name**, a Max Run Time and a max runtime failure status if desired. Click the **Save** button to save the check point.



to save the test.

## Creating a Database Test

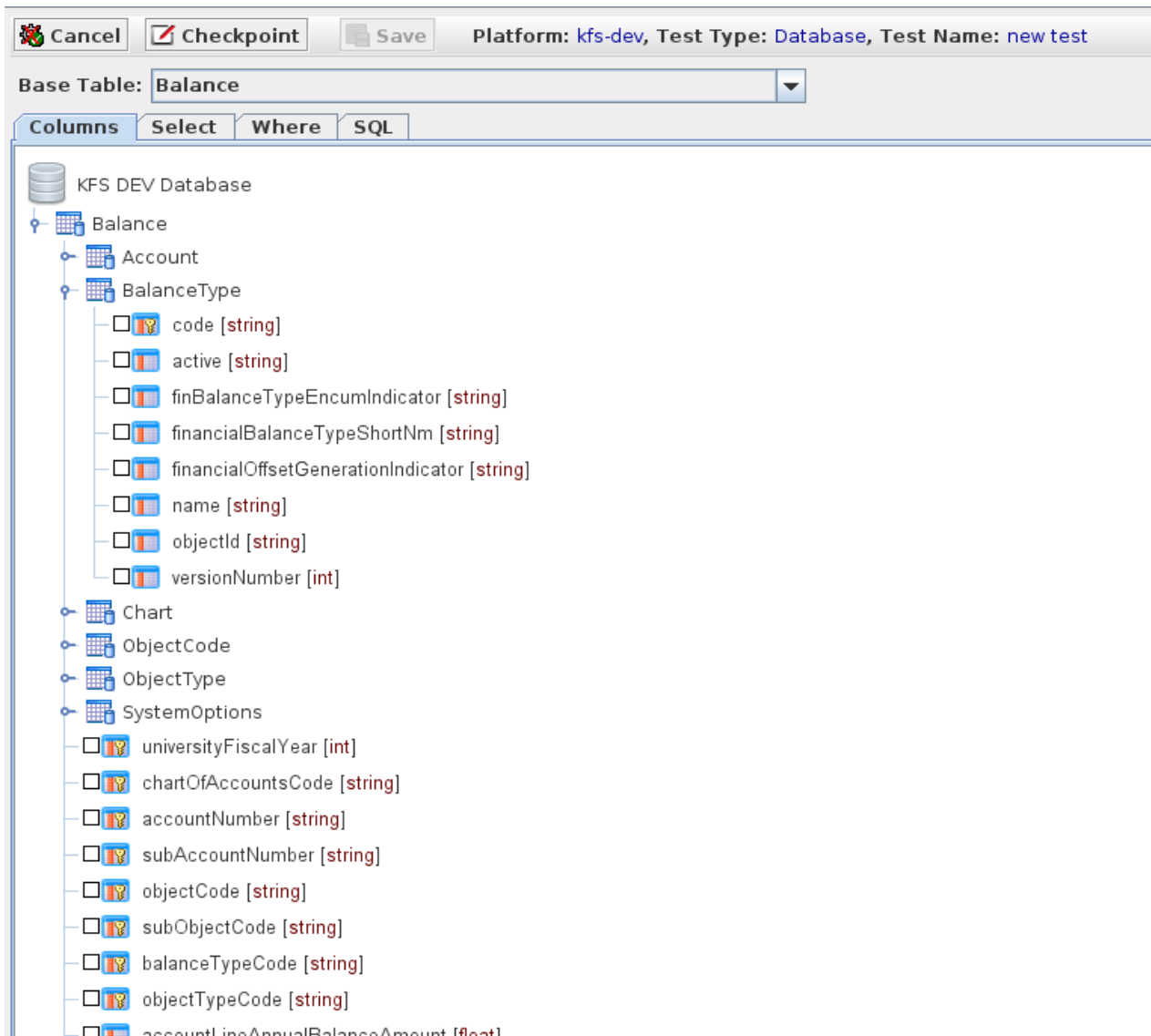
Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Database** is selected from the drop down. Click **Continue** to load the initial database test setup panel.



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.



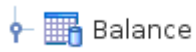
## Selecting Columns for Database Query

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:

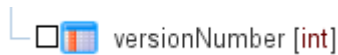
### Database associated with test platform



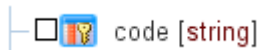
## Database Table



## Table Column

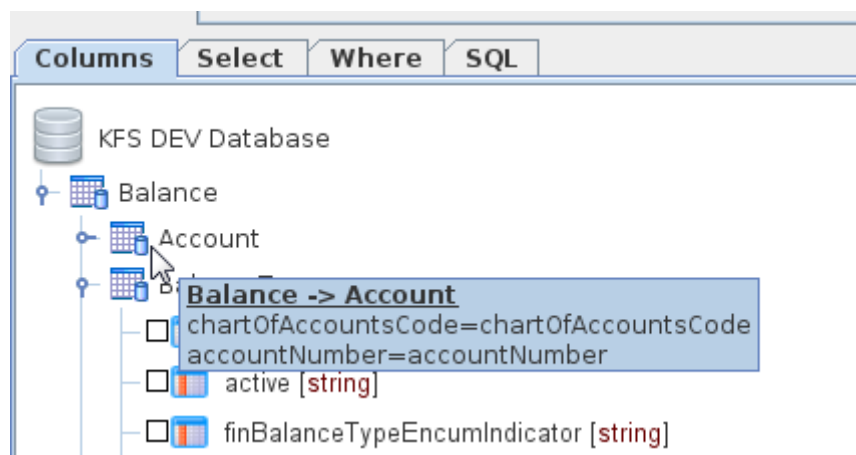


## Primary Key Table Column

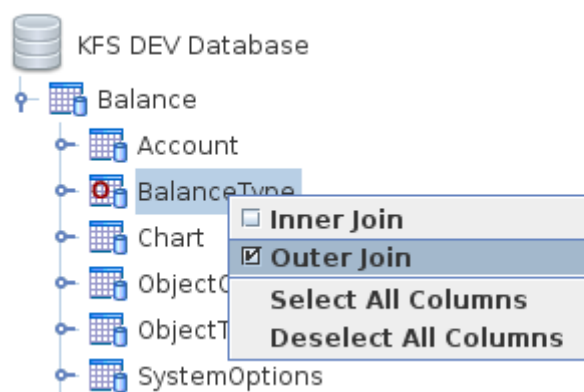


Column data type is displayed in square brackets: [string]

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:

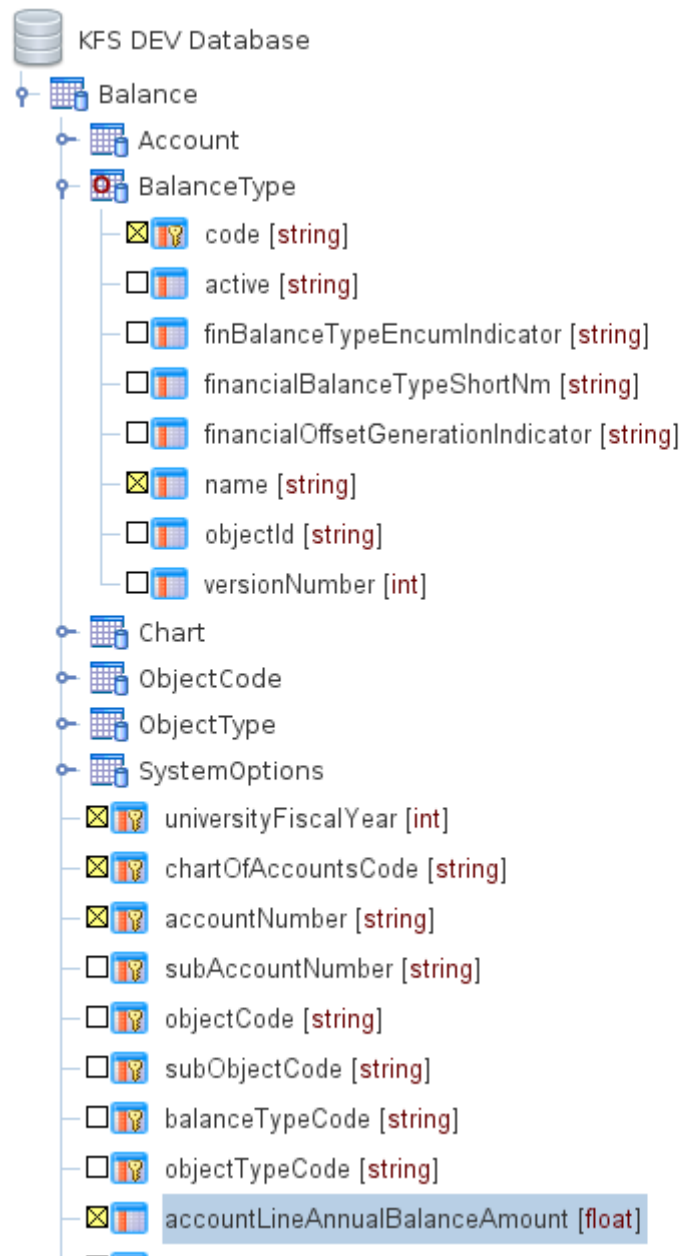


By default multi-table column selections will create an inner-join query. If an outer join is desired, right click on the table desired and select the outer join option:



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.

table	column	function	order	asc/desc
Balance	accountLineAnnualBalanceAmount	float	1	ASC DESC

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

Select Columns
<input type="checkbox"/> Add Column <input type="checkbox"/> 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:

Base Table:	Columns	Select	Where	SQL
Balance				

and/or	(	table	column	operator	value	)

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

Columns	Select	Where	SQL

Where Comparisons														
<input type="button" value="+ Add Comparison"/> <input type="button" value="X Delete Comparison"/>														
<table border="1"> <thead> <tr> <th>and/or</th> <th>(</th> <th>table</th> <th>column</th> <th>operator</th> <th>value</th> <th>)</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td>Balance</td> <td>BalanceType</td> <td>=</td> <td></td> <td></td> </tr> </tbody> </table>	and/or	(	table	column	operator	value	)			Balance	BalanceType	=		
and/or	(	table	column	operator	value	)								
		Balance	BalanceType	=										

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

Columns	Select	Where	SQL
Where Comparisons			
<input type="button" value="+ Add Comparison"/> <input type="button" value="X Delete Comparison"/>			
and/or	(	table	column
		Balance	accountLineAnnualBalanceAmount [float]
			operator
			>
			value
			100000
			)

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

Columns	Select	Where	SQL
Where Comparisons			
<input type="button" value="+ Add Comparison"/> <input type="button" value="X Delete Comparison"/>			
and/or	(	table	column
		Balance	accountLineAnnualBalanceAmount [float]
			operator
			>
			value
			100000
			)
AND	(	BalanceType	code [string]
			operator
			=
			value
			code1
OR		BalanceType	code [string]
			operator
			=
			value
			code2
			)

## Displaying Generated SQL and Verifying Validity

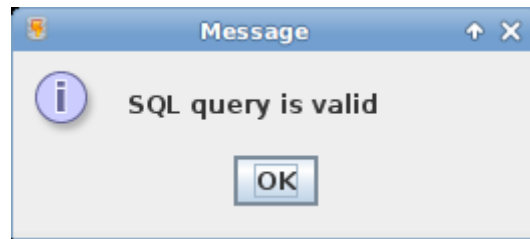
To view and validate the SQL statement that will be generated, click the **SQL** tab.

Columns	Select	Where	SQL
<input type="button" value="Check Generated SQL"/>			
<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>			

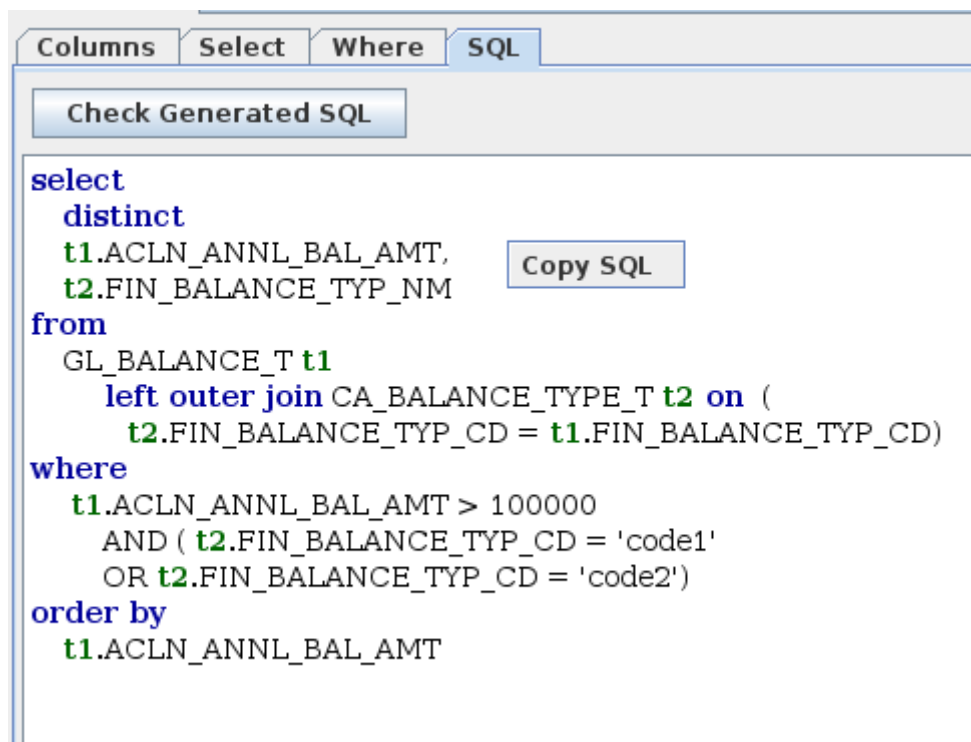
Click the **Check Generated SQL** button



to verify the SQL.

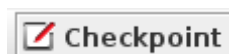


Right click on the SQL display screen and click the **Copy SQL** popup menu item to copy the generated SQL to the clipboard.



### ***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:

Checkpoint Property:

On Failure:

Max Run Time (sec):

☐ Save SQL Query Results

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

Checkpoint Property:

On Failure:

Max Run Time (sec):

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

On Failure:

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

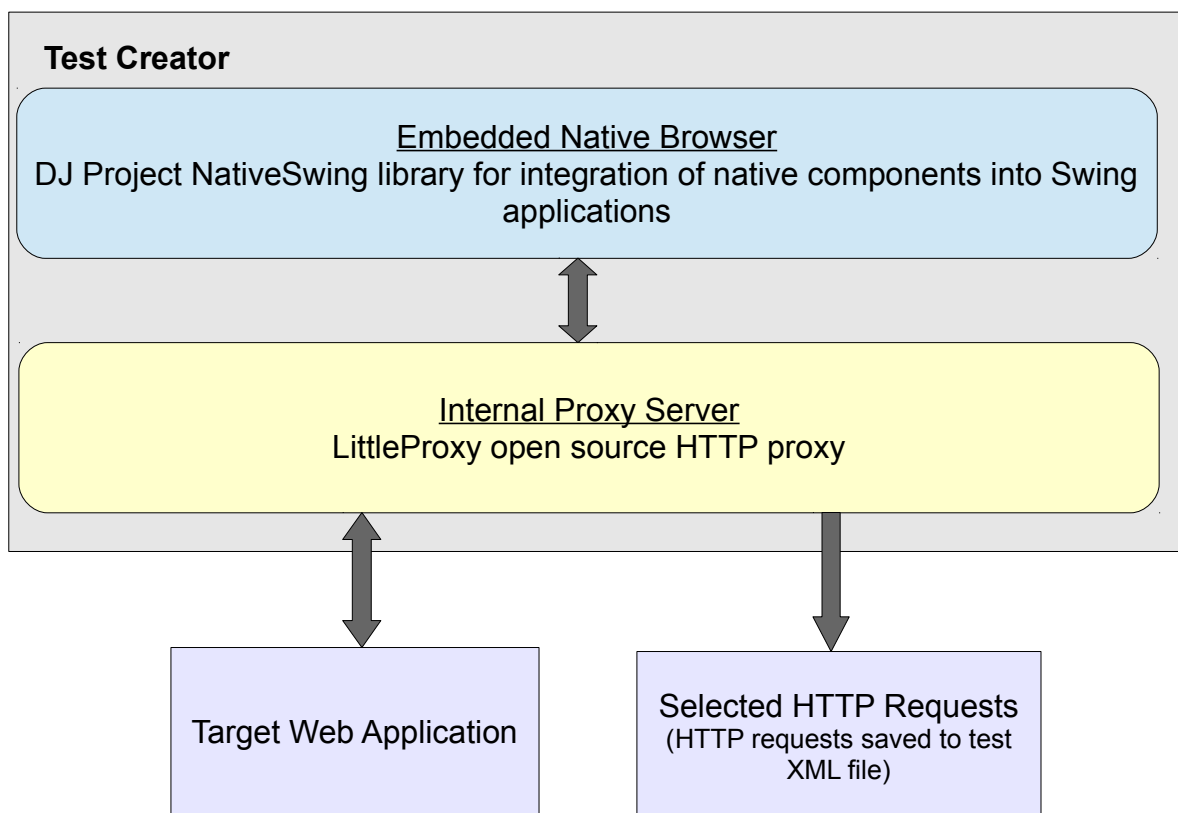
Pull up the **Initialize New Test** dialog as described in the **Creating a New Test** section and complete the entries as desired. Ensure that the **Test Type** of **Web** is selected from the drop down. Click **Continue** to load the initial database test setup panel.

### *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:

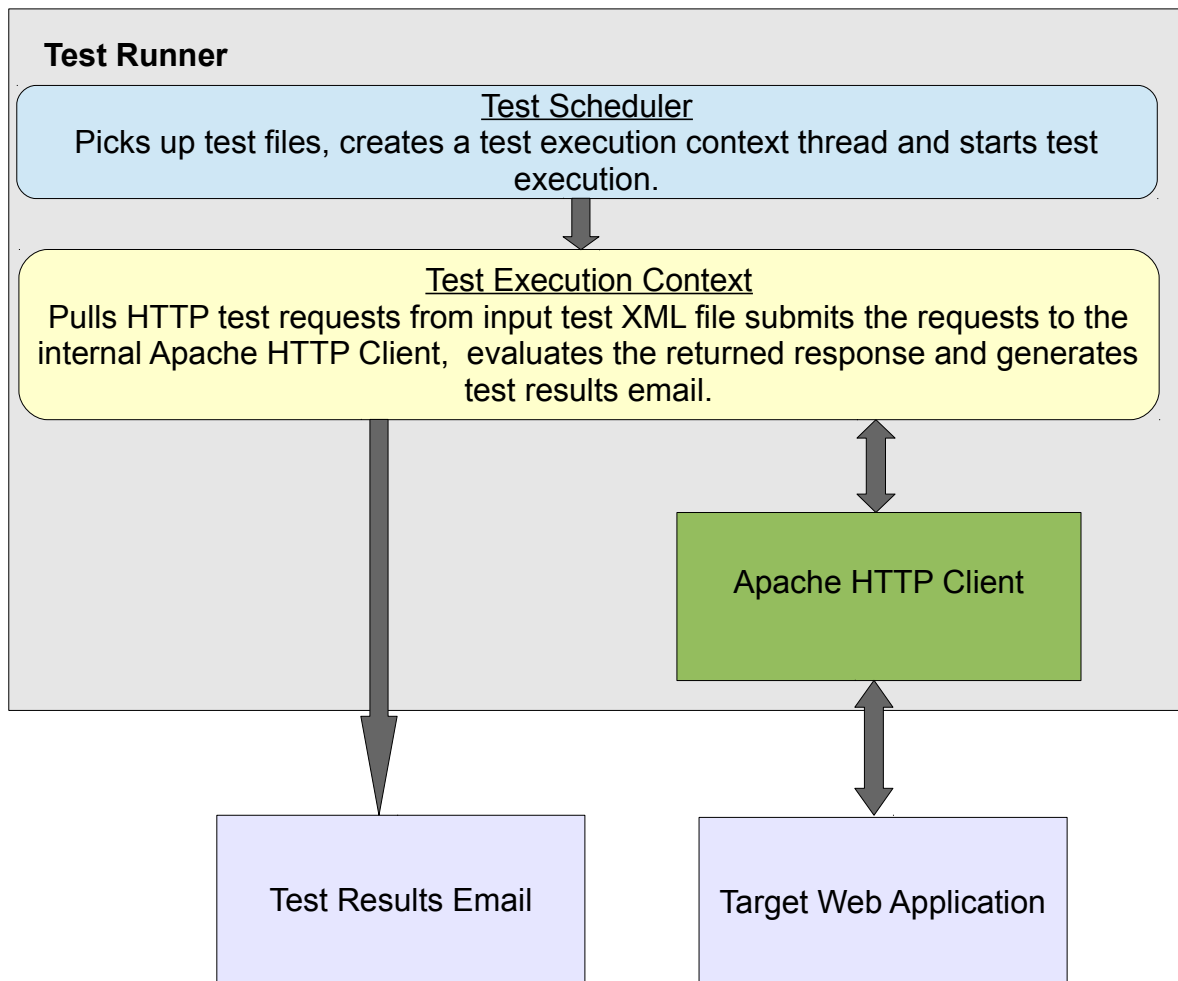


The proxy server allows the application to act as a “man-in-the-middle” and grab non encrypted HTTPS requests before they are sent to the target web application. To use the proxy server the test creator must be started with the appropriate input parameters as

described in section ***Test Creator Command Line.***

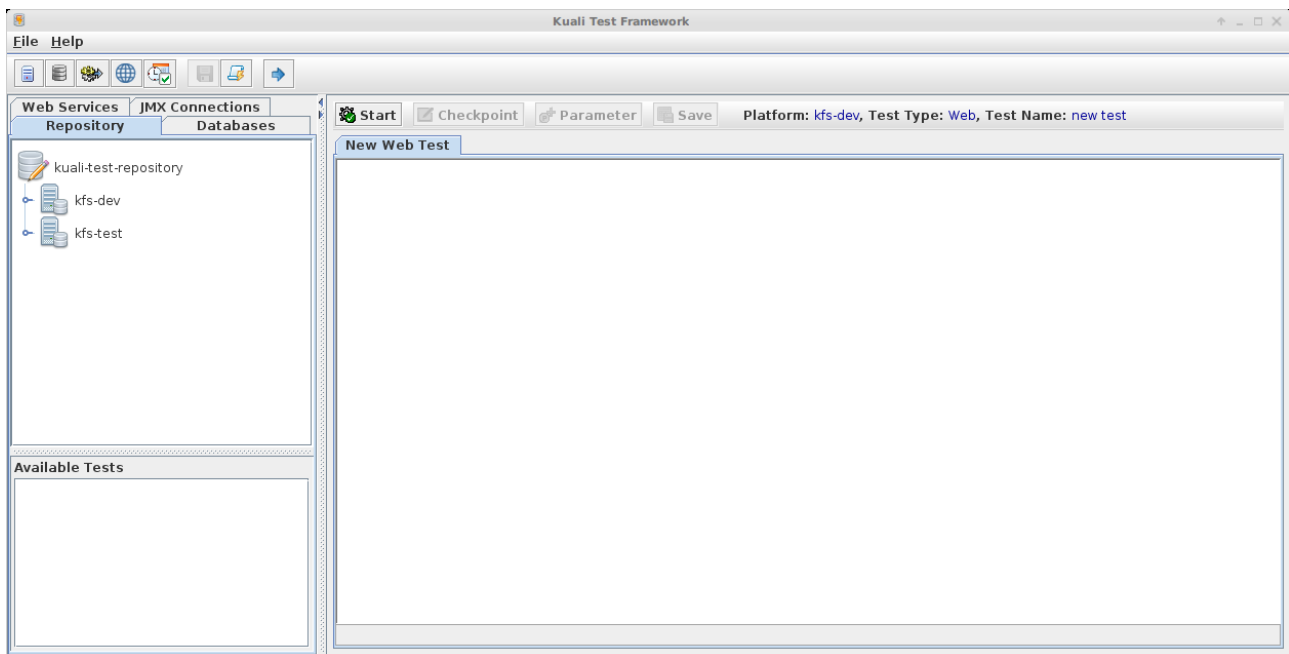
## Test Runner

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:



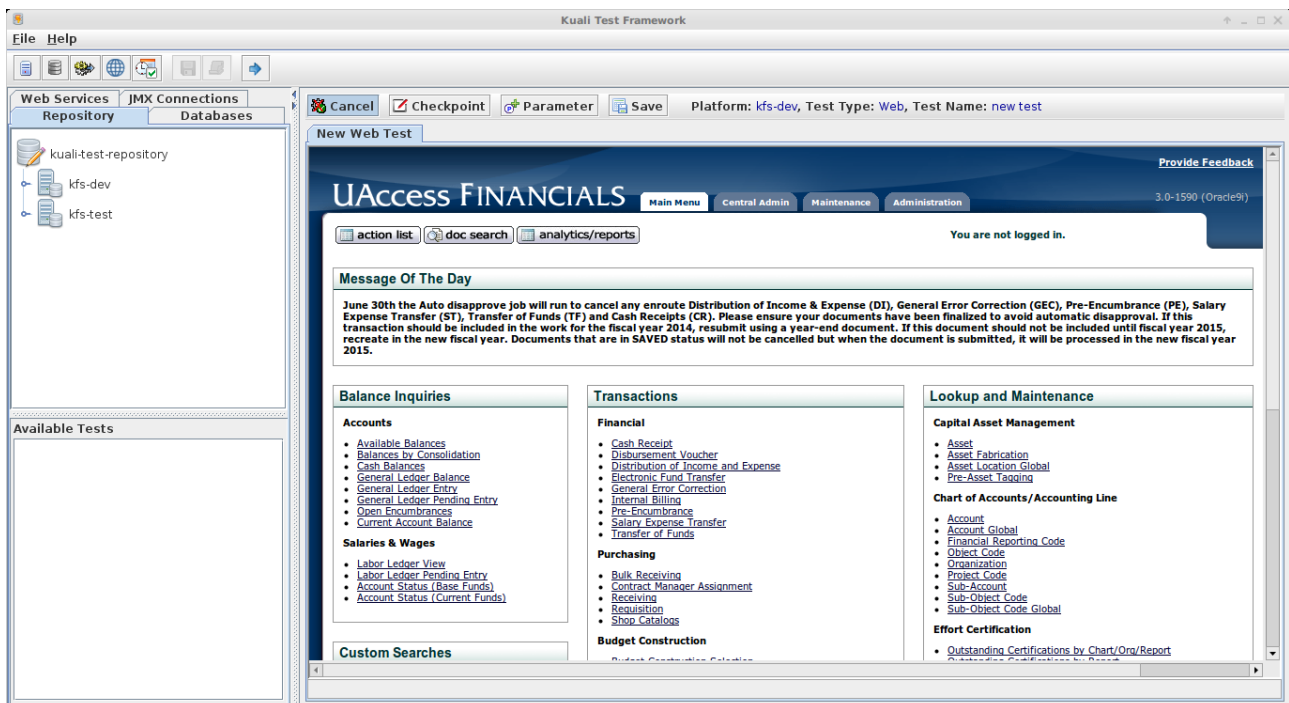
## Starting a Web Application Test

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

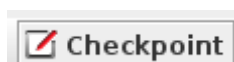


Click the Start button  to begin the test.

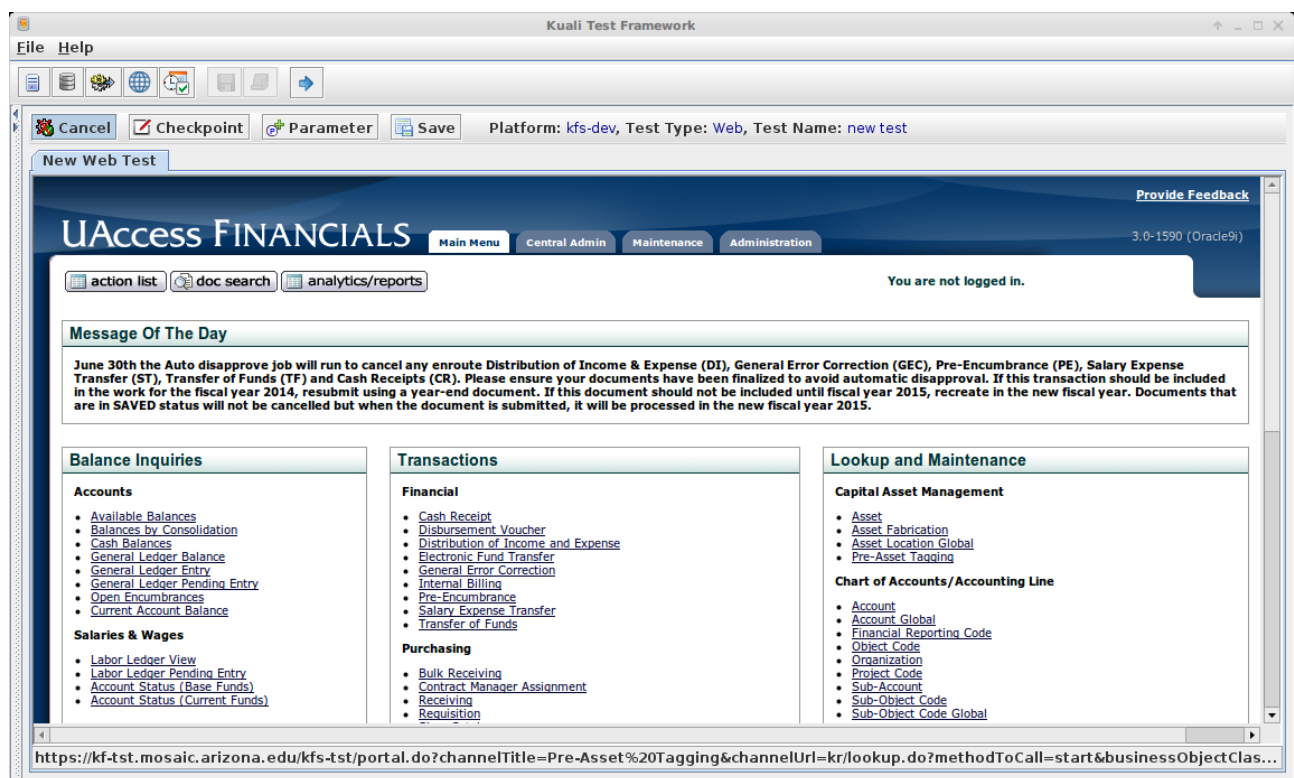
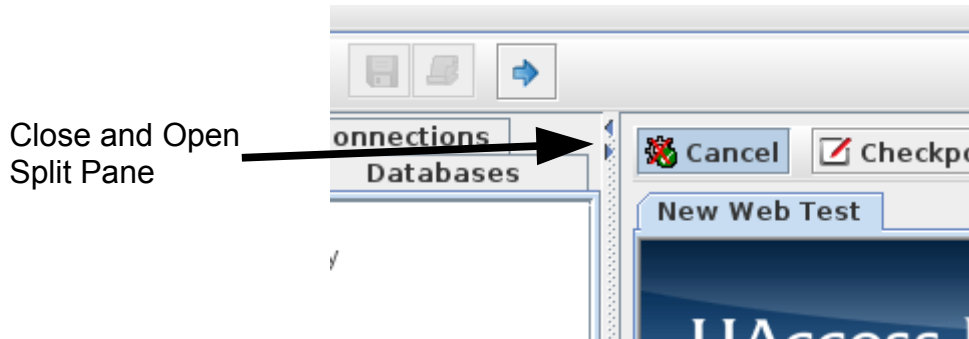
Once the web test is initialized the default native browser should display in the web test panel:



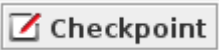
Now you can navigate through the target web application as normal. At any point you can click the Checkpoint button

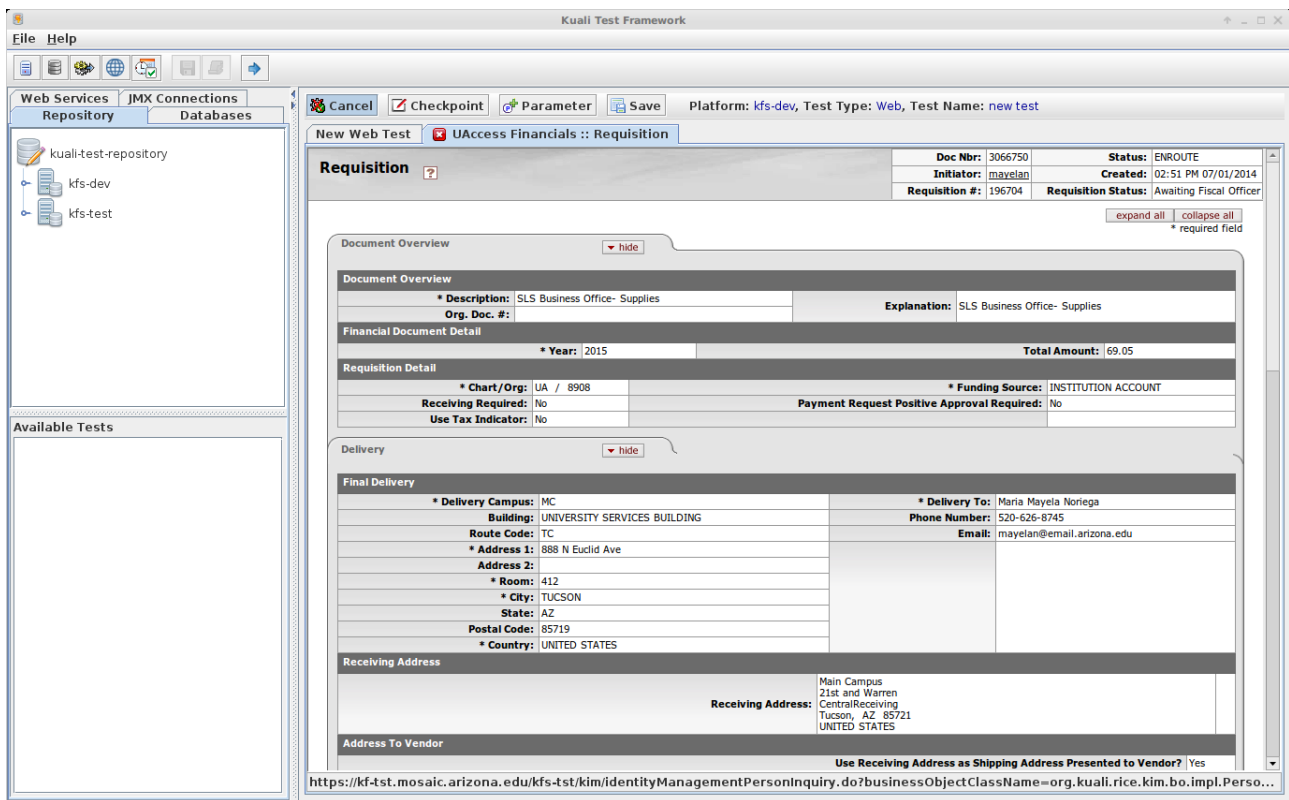


to create a check point that will be used as evaluation criteria during test runs. To better see the full web page click the split pane icon shown below – this will close the management pane:

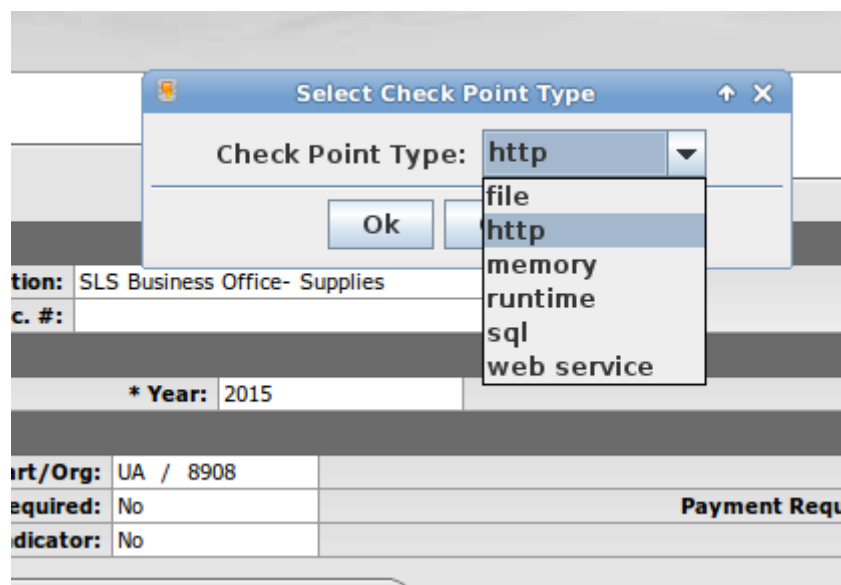


## Creating a Web Checkpoint

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.

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. Tag handler configuration will be discussed in detail later in this document but a couple of 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

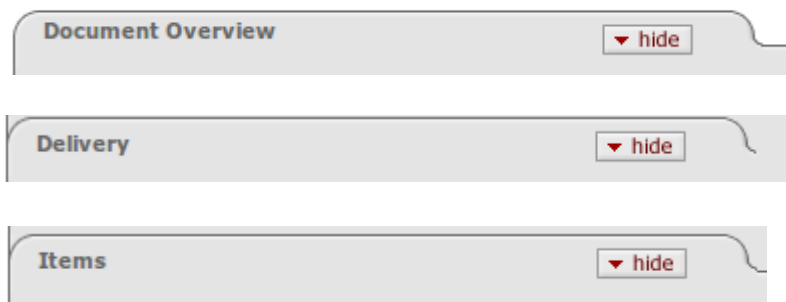
```

<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	DocumentOverview	Header Info	Items	PaymentInfo	RouteLog	Vendor
AccountSummary		AdHocRecipients				AdditionalInstiti
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:		equal to		
<input type="checkbox"/>	Vendor Address	Address 2:		equal to		
<input type="checkbox"/>	Vendor Address	Postal Code:		equal to		
<input type="checkbox"/>	Vendor Address	Attention:		equal to		
<input type="checkbox"/>	Vendor Address	Country:		equal to		
<input type="checkbox"/>	Vendor Info	Customer #:		equal to		
<input type="checkbox"/>	Vendor Info	Notes To Vendor:		equal to		
<input type="checkbox"/>	Vendor Info	Payment Terms:		equal to		
<input type="checkbox"/>	Vendor Info	Shipping Title:		equal to		
<input type="checkbox"/>	Vendor Info	Shipping Payment Te...		equal to		
<input type="checkbox"/>	Vendor Info	Contract Name:		equal to		
<input type="checkbox"/>	Vendor Info	Contacts:		equal to		
<input type="checkbox"/>	Vendor Info	Phone Number:		equal to		
<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 shows the Kuali Test Framework interface. On the left, there's a sidebar with 'Web Services' and 'JMX Connections' tabs. The 'Repository' tab is active, showing a tree view with 'kuali-test-repository', 'kfs-dev', and 'kfs-test'. Below this is the 'Available Tests' section. The main area displays the 'New Web Test' configuration. At the top, there's a toolbar with 'Cancel', 'Checkpoint', 'Parameter', and 'Save' buttons. The 'Platform' is 'kfs-dev', 'Test Type' is 'Web', and 'Test Name' is 'new test'. The 'UAAccess FINANCIALS' application is loaded, showing the 'Requisition' form. The 'Document Overview' section includes fields for 'Description', 'Org. Doc. #', and 'Explanation'. The 'Financial Document Detail' section includes fields for 'Year' (2015), 'Chart/Org' (UA / 9507), 'Funding Source' (INSTITUTION ACC), 'Receiving Required' (checkbox), and 'Payment Request Positive Approval Required' (checkbox). The 'Delivery' section includes fields for '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		<a href="#">logout</a>	
	<b>Doc Nbr:</b>	3068813	
	<b>Initiator:</b>	rbtucker	
	<b>Requisition #:</b>	Not Available	<b>R</b>

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:

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
accountNumber	
chartOfAccountsCode	
docId	
documentNumber	
documentTypeId	
documentTypeName	
financialObjectCode	
fiscalYear	

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**

**DocumentOverview** **Header Info** **Items** **PaymentInfo** **RouteLog** **Vendor**

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

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

**Add new test execution parameter**

Parameter Name:

Parameter Value:

**Parameters**

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

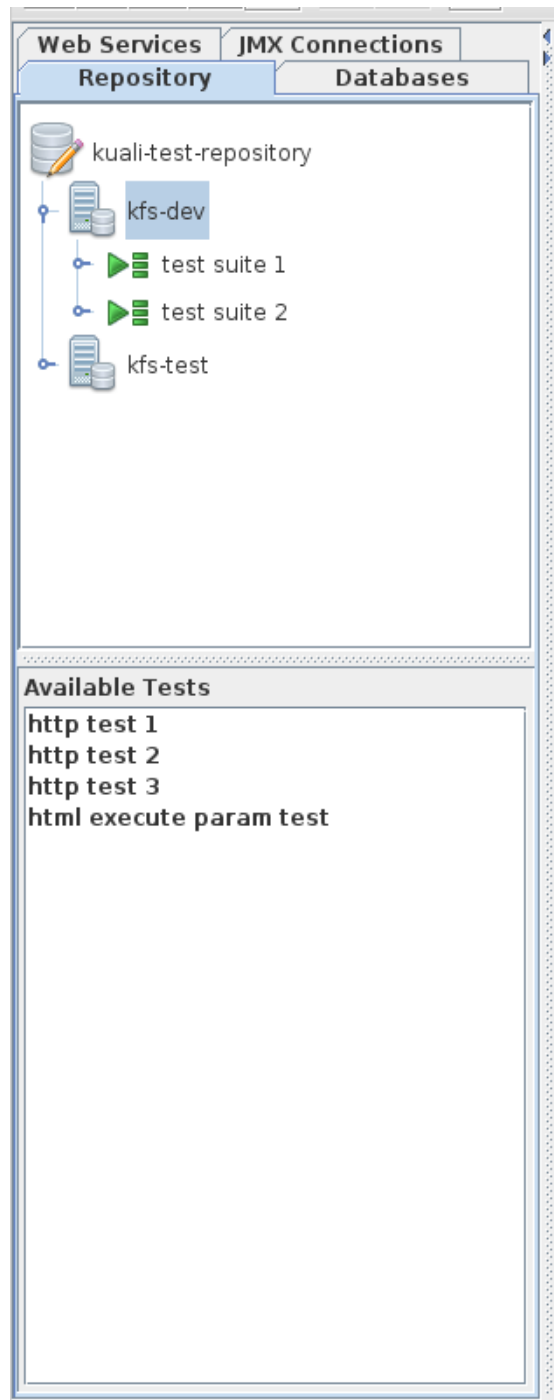


select the parameter you wish to remove from the **Parameters** table, click the **Remove Parameter** button then click **Save**.

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 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**:

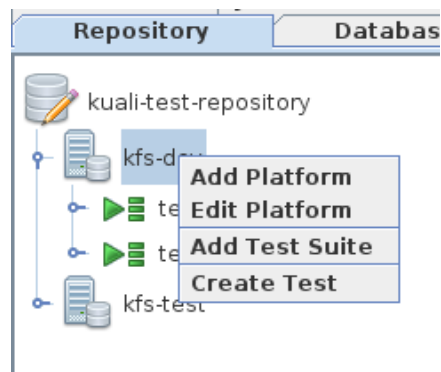


## Repository

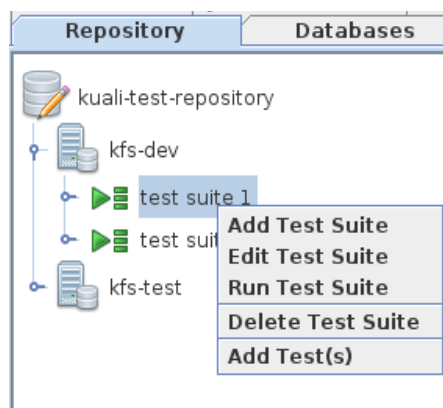
The repository tab is a logical representation of the test repository. The upper pane contains a tree view of the test repository. The lower pane lists available platform tests for a selected platform. Most functionality is driven by a popup menu. Several of which are shown below. When a test is created 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.



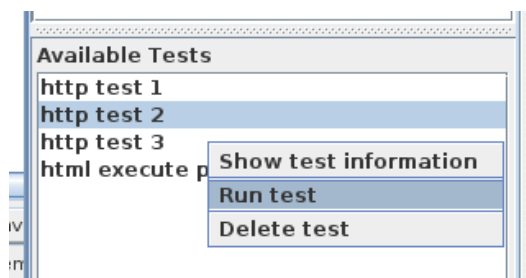
## Platform Popup Menu



## Test Suite Popup Menu

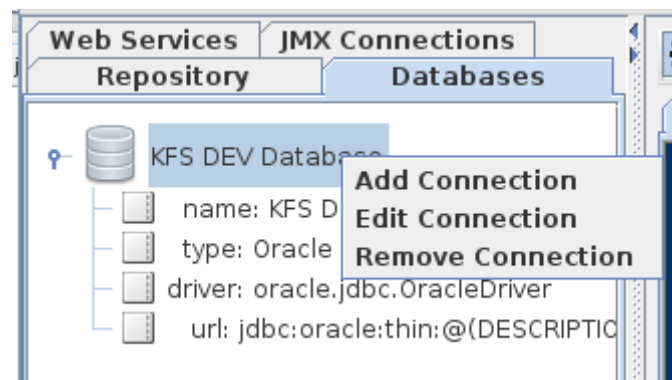


## Test Popup Menu



## 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.

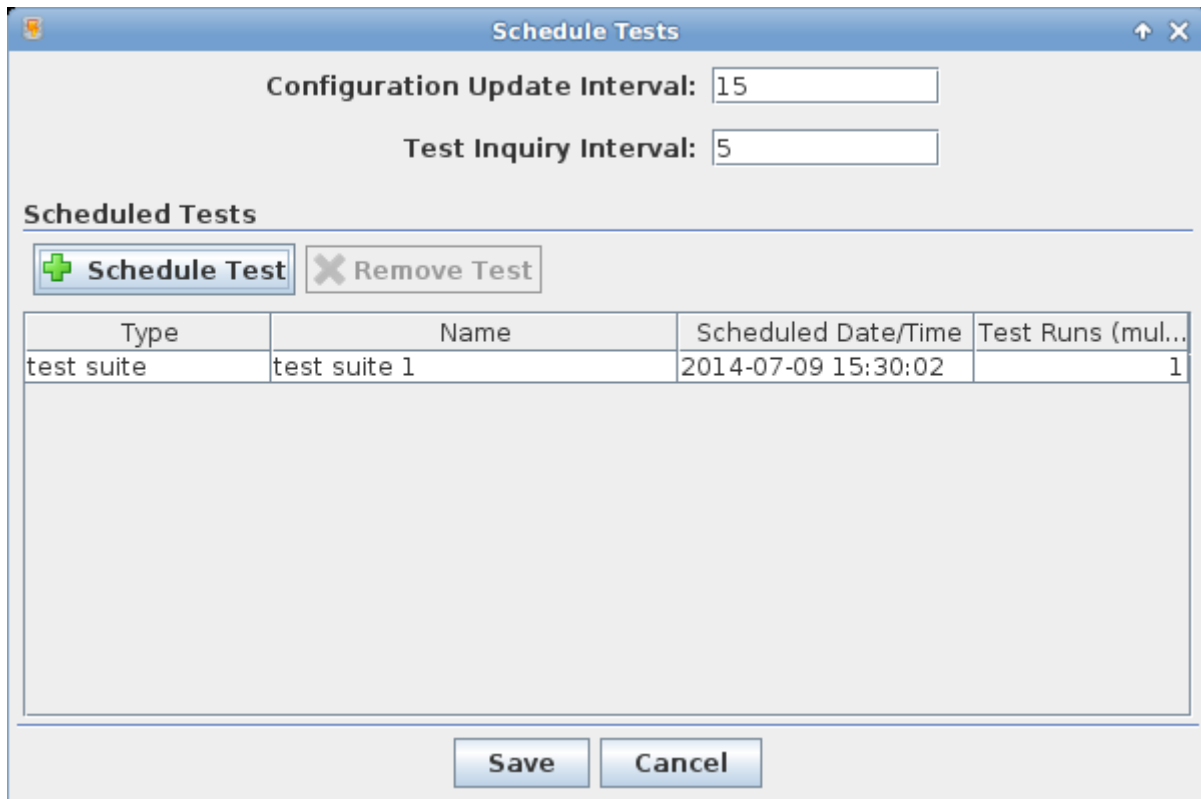


## Test Scheduling

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.

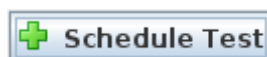
A screenshot of the 'Schedule Tests' dialog box. It has a title bar with a maximize button and a close button. The dialog contains two input fields: 'Configuration Update Interval: 15' and 'Test Inquiry Interval: 5'. Below these is a section titled 'Scheduled Tests' which contains two buttons: '+ Schedule Test' and 'X Remove Test'. Underneath 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 two buttons: 'Save' and 'Cancel'.

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

## Tag Handler Configuration Detail

Full details of tag handler XML configuration will go here.