EMC® Documentum® Web Development Kit Automated Test Framework

Version 2.5

User Guide

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Published August 2008

Table of Contents

List of Figures

List of Tables

Preface

This document describes how to use the EMC Documentum Web Development Kit Automated Test Framework.

Intended Audience

The audience of this manual is intended to be quality assurance engineers and developers who want to use the WDK Automated Test Framework and who are familiar with WDK.

Revision History

The following changes have been made to this document.

Revision History

Revision Date	Description
August 2008	Initial release for version 2.5.

Chapter 1

Overview

These topics are included:

Introduction

The Web Development Kit (WDK) Automated Test Framework helps automate the generation of and running reusable test cases for WDK applications. You can use the WDK Automated Test Framework to perform these kinds of tests on WDK applications:

- Sequence of user actions
- Validate actions
- Validate the states of controls

To ensure reusable test cases, the WDK Automated Test Framework does not depend on nor test the locations of user interface elements on the browser page.

Tasks

You use the WDK Automated Test Framework to perform these tasks:

- Create reusable test cases (see To create a reusable test case for a component or action:, page 10.
- Execute test cases (see Chapter 5, Executing Test Cases and Test Suites)
- Troubleshoot test case failures and errors (see Chapter 6, Understanding Test Results)
- Create unit tests (see Chapter 9, Implementing unit tests for components)

You can take advantage of new features without having to re-record you existing test cases. See Chapter 10, Modifying Existing Test Cases to Use New Features.

To create a reusable test case for a component or action:

1. Define (for example, in a test specification) your test case.
2. Use the **Test Recorder Launcher** to record the test case. See Chapter 3, Recording Test Cases.
3. Play back your test case using the **Test Launcher**. See Chapter 5, Executing Test Cases and Test Suites.
4. Make it reusable. See Chapter 4, Making Test Cases Reusable.
5. Debug the test case to make sure the test case runs as expected. See Chapter 5, Executing Test Cases and Test Suites and Chapter 6, Understanding Test Results.

For more information

For an introduction to the Documentum system, see the *Content Server Fundamentals*.

For information about configuring and customizing WDK and WDK applications, see the *Web Development Kit Development Guide*. For information about deploying WDK and WDK applications, see the *Web Development Kit and Webtop Deployment Guide*.

Chapter 2

Installation and Configuration

These topics are included:

Introduction

You install the WDK Automated Test Framework by unzipping its files onto an existing Webtop installation.

Installation procedure

To install the WDK Automated Test Framework:

1. Install Webtop 6.5 on a supported application server. For a list of supported application servers, see the *Webtop Release Notes* Version 6.5. For installation instructions, see the *Web Development Kit and Webtop Deployment Guide* Version 6.5.
2. Stop the application server.
3. Unzip the WDK_TestFramework_2.5.zip file into the application server's Webtop web application folder.
4. Restart the application server.

Configuring the WDK Automated Test Framework

You configure the WDK Automated Test Framework by specifying custom elements and attributes in a configuration file that extends the testframeworkconfig id="default" definition in the webtop/config/test/testframework_config.xml file. The configurable elements are described in the following list:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<config version='1.0'>
  <scope>

    <testframeworkconfig id="default" extends:"default:webtop/config/test/
                                                testframework_config.xml">

         <!-- enter 0 to n listeners -->
         <eventlisteners>
            <class>com.documentum.web.test.TestResultLogger</class>
         </eventlisteners>

         <!--  the testcase navigator -->
         <testcasenavigator>
            <class>com.documentum.web.test.TestCaseNavigator</class>
         </testcasenavigator>

      1<clienteventdelay>1000</clienteventdelay>
      2<testcasetimeout>10</testcasetimeout>
      3<importclasses>
            <class>some class</class>
        </importclasses>

        <!-- Event generators leveraged by the SetValueEvent class -->
        <setvalueeventgenerators>
           <class>com.documentum.web.test.controlevent.dmf.TextSetValueEvent</class>
            ...
        </setvalueeventgenerators>

        <!-- Event generators leveraged by the ClickEvent class -->
        <clickeventgenerators>
           <class>com.documentum.web.test.controlevent.dmf.ButtonClickEvent</class>
           ...
        </clickeventgenerators>

        <!-- Event generators leveraged by the MouseOverEvent class -->
        <mouseovereventgenerators>
           <class>com.documentum.web.test.controlevent.dmf.MenuMouseOverEvent</class>
           <class>com.documentum.web.test.controlevent.dmfx.
                  MoreActionMenuItemMouseOverEvent</class>
        </mouseovereventgenerators>

      4<teststepactionvalidators>
            <entry>
               <action>*</action>
```

```
            <class>com.documentum.web.test.validation.DefaultValidator</class>
         </entry>
      </teststepactionvalidators>

      <!-- ITestStepActionValidationProducer implementations called during
           action recording -->
      <teststepactionvalidationproducers>
      <!--<entry>-->
         <!--<action>actionid</action>-->
         <!--<class>fully.qualified.path.to.ITestStepActionValidationProducer.
                   implementation</class>-->
      <!--</entry>-->
         <entry>
            <action>*</action>
            <class>com.documentum.web.test.validation.DefaultValidationProducer</class>
         </entry>
      </teststepactionvalidationproducers>

      <!-- Event producers (output event in the format of java statement)
           leveraged by TestStepClassProducer during recording.
           Control event is a server-side event triggered by a control; e.g.
           "onButtonClick" event for the following control:
           <dmf:button onclick='onButtonClick'>                          -->
      <controleventproducers>
         <class>com.documentum.web.test.controlevent.UnnamedControlEvent</class>
         ...
      </controleventproducers>

      <!-- Event producers (output event in the format of java statement)
           leveraged by TestStepClassProducer during recording.
           Client user event is a client-side event triggered by a control; e.g.
           "onOK" event for the following control:
           <dmf:button onclick='onOK' runatclient='true'>    -->
      <clientusereventproducers>
         <class>com.documentum.web.test.recorder.ClientSideControlEventProducer</class>
         ...
         </clientusereventproducers>

      <!--
         ClientUserEventCreators implement the IClientUserEventCreator
         interface and are used to generate specialized versions of
         ClientUserEvent during recording.

         If you require a specialized ClientUserEvent (for example,
         to store additional state) you may register your implementation of
         IClientUserEventCreator by adding it to the list below.

         The first class which returns true for IClientUserEventCreator.canCreate
         will be used to generate the specialized ClientUserEvent instance,
         when hoarding events in ControlEventManager/ClientUserEvent.makeEvent
      -->
      <clientusereventcreators>
         <class>com.documentum.web.test.recorder.TreeNodeUserEvent</class>
      </clientusereventcreators>

5<ignorableserverevents>
         <event>onComponentJump</event>
```

```
        ...
   </ignorableserverevents>

6<validargumentnames>
        <argument>objectId</argument>
        ...
   </validargumentnames>

7<componentframemap>
        <entry>
           <component>titlebar</component>
           <frame>titlebar</frame>
        </entry>
        ...
    </componentframemap>

 8<staticcomponents>
        <component>titlebar</component>
        ...
   </staticcomponents>

 9<ignorablecontrolevents>
        <entry>
           <component>statusbar</component>
           <control>tabsView</control>
        </entry>
        ...
    </ignorablecontrolevents>

 10<ignorablecontroltypes>
        <class>com.documentum.web.formext.control.action.ActionMultiselect</class>
        ...
    </ignorablecontroltypes>

 11<objectidvariablenames>
        <name>objectId</name>
        ...
    </objectidvariablenames>

 12<ignorableattributenames>
        <attribute>i_vstamp</attribute>
        ...
    </ignorableattributenames>

 13<ignorableforms>
        <class>fully.qualified.class</class>
        ...
    </ignorableforms>

   <!-- Provides support for user-defined classes and parameter for
        dealing with object Ids. -->
    <ididentifierfunctionproviders>

        <!-- an example, can be replaced by users -->
        <provider>
           <type>dm_sysobject</type>
           <class>com.documentum.web.test.IdIdentifierSysObjectProvider</class>
```

```
      </provider>

      <!-- an example, can be replaced by users -->
      <provider>
         <type>dm_acl</type>
         <parameters>
            <param>
               <name>name</name>
               <attribute>object_name</attribute>
            </param>
         </parameters>
      </provider>
      ...
   </ididentifierfunctionproviders>

   <treenodeidentifierfunctionproviders>
      <provider>
         <type>com.documentum.web.form.control.Tree</type>
         <class>com.documentum.web.test.TreeNodeIdentifierFunctionProvider</class>
      </provider>
   </treenodeidentifierfunctionproviders>

   <enableinspectionmode>true</enableinspectionmode>

   <!-- these controls should not be automatically surrounded by inspection tags -->
   <inspectionresistantcontrols>
      <control>com.documentum.web.form.control.Option</control>
      ...
   </inspectionresistantcontrols>

14 <validdatafields>
      <identitydatafield>r_object_id</identitydatafield>
   </validdatafields>

15 <uivalidation>
   16 <completetestonfailure>true</completetestonfailure>
   17 <localeagnosticrecording>
   18 <menueventdelay>
      <controls>
         <control>
            <class>*</class>

            <definitioncomponent>inspector</definitioncomponent>
            <fieldgetters>
               <field name="visible" getter="isVisible" />
               <field name="enabled" getter="isEnabled" />
               <field name="name" getter="getName" />
               <field name="tooltip" getter="getToolTip" />
               <field name="rowIndex" getterclass="com.documentum.web.test.validation.ui.
                           fieldhandler.RowIndexFieldHandler" />
               <field name="databoundContainerName" getterclass="com.documentum.web.test.
                      validation.ui.fieldhandler.DataboundContainerNameFieldHandler" />
               <field name="datafield" getter="getDatafield" />
            </fieldgetters>
            <validator>com.documentum.web.test.validation.ui.DefaultUIValidator</validator>
         </control>
         ...
```

```
            </controls>
          </uivalidation>
        </testframeworkconfig>
    </scope>
</config>
```

1 clienteventdelay causes every client event that the test case processes to be wrapped by an instance of DelayClientEvent with the delay time (in milliseconds) specified in this element. Events will be executed with the specified delay between each event instead of practically simultaneously.
2 testcasetimeout specifies the amount of time to pass without any activity in a test step, after which the test case fails and the next test case is executed. After the last test case, the results are displayed.
3 importclasses is used to specify additional classes to be imported in test case Java classes.
4 teststepactionvalidators specifies the fully qualified validator classes (that is, ITestStepActionValidator implementations) for actions. The action element specifies the value of the ID attribute of the action element in the action's configuration definition. The class element specifies the fully qualified name of a class that handles validation.
5 ignorableserverevents specifies server events (non-user-triggered events on non-controls, such as onComponentJump) to be ignored (that is, not recorded).
6 validargumentnames specifies the arguments, which are enclosed by databound control tags, to be used to resolve the control. When an event is recorded, the valid arguments (if the exist) are stored in the event together. The event producer can then use it to produce a databound event.
7 componentframemap specifies the name of the frame in which a component resides. Control events generated for controls located in these components are wrapped by a FrameClientEvent targeting the specific frame.
8 staticcomponents specifies components that persist throughout the entire user session, so that the information of the frame in which it resides is always known by the test recorder.
9 ignorablecontrolevents specifies the control events to ignore when the control events are triggered by client events. Consequently, instead of recording both the client event and control event, only the client event is recorded.
10 ignorablecontroltypes specifies the fully qualified classes of the controls to ignore during recording (for example, com.documentum.web.form.control.Hidden).
11 objectidvariablenames specifies that parameters that are supposed to be an object ID are to be written out as a test case variables in an IdIdentifier format in the test case's configuration file.
12 ignorableattributenames specifies attributes (such as, the modified date attribute that changes every time a test case is executed) that are not to be validated.
13 ignorableforms specifies the fully qualified class of a form to be ignored during recording. For example, you would not want to record test tool-related components, which are not part of your application.
14 validdatafields specifies the name of the datafield, which a control uses, that is used to resolve the control during playback.
15 uivalidation specifies some user interface-specific options as well as the control fields to validate. For each control to validate, you specify the fields in field elements.
The elements for control are:

- class: Fully qualified class name of the control.
- fieldgetters: Contains field elements.
- validator: Fully qualified name of a class that handles validation. For example, the com.documentum.web.test.validation.ui.DefaultUIValidator interface.

The attributes for field elements are:

- name: Logical name of the control field to validate.
- getter: Method name. You can specify getter or getterclass.
- getterclass: Fully qualified class name that is an implementation of the IUIValidationFieldHandler interface. You can specify getter or getterclass.

To enable control state validation, set the enableinspectionmode element to true.

16 To continue executing test cases even after a user interface validation failure and report them after test case execution completes, set <completetestonfailure> to true.

You should set this option to true when you do not want to terminate test execution after encountering the first user interface validation failure.

17 To use the underlying value instead of locale-specific values, set <localeagnosticrecording> to true.

During test execution, Producers for DropDownListEvents and ListBoxEvents use the underlying value instead of writing out locale-specific labels as choices to select.

You should set this option to true when you want to ignore locale-specific values.

18 To set action menu item click events to be delayed, specify the desired number of milliseconds in <menueventdelay>.

You should set this option when the menu takes a long time to render, because if an ActionMenuItem's click JavaScript event fires immediately after clicking a menu, the menu item might not be found.

Chapter 3

Recording Test Cases

These topics are included:

Overview

Recording a test case generates skeleton classes and XML configuration files for the component being tested. To create a reusable test case for the component, you must implement certain interfaces in the skeleton classes and configure the test case in the XML files.

General procedure for recording test cases

To record a test case:

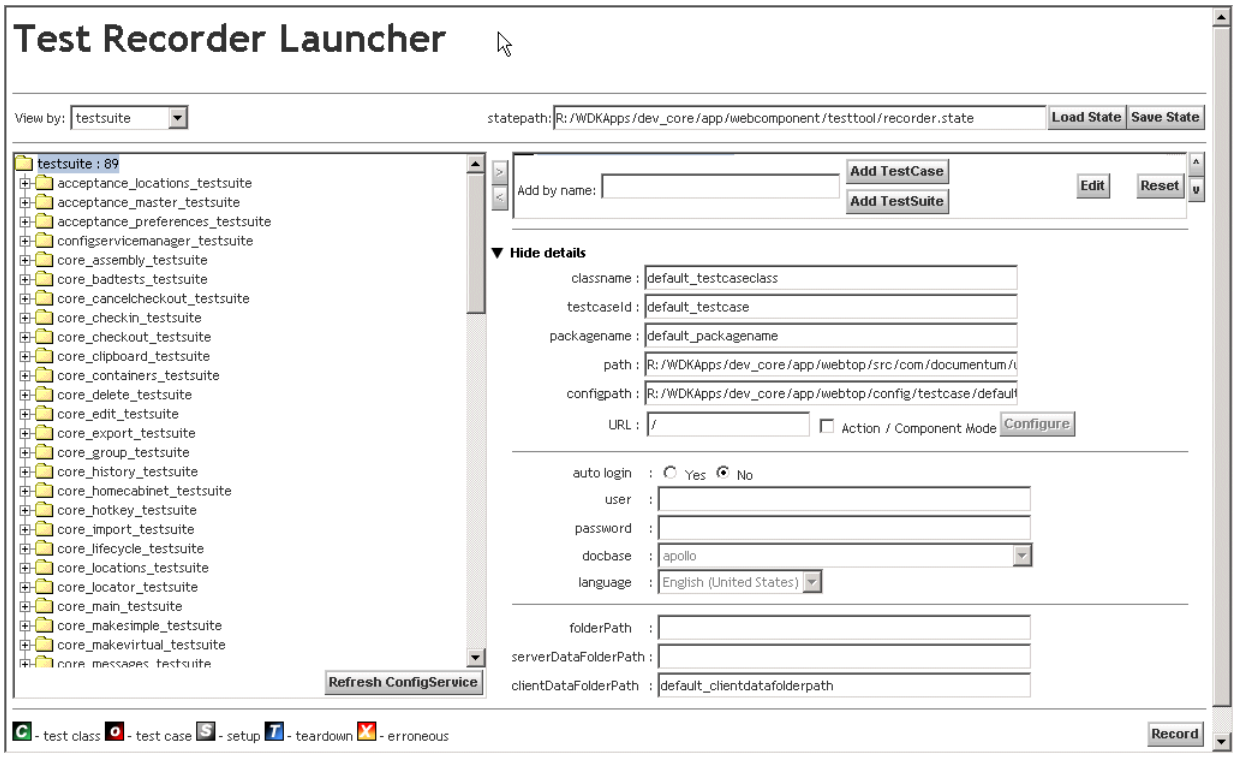
1. Set options in your testframework_config.xml file as described in Configuring the WDK Automated Test Framework, page 12.
2. Launch the **WDK Test Framework Tools** page by specifying this URL:

   `http://host:port/wdk-app/component/testtool`

   where:
   - *host*: Name (or IP address) of the machine on which the WDK Automated Test Framework is installed.
   - *port*: Port number at which the WDK Automated Test Framework listens.
   - *wdk-app*: Name of the WDK application that you have installed. Default is webtop.
3. Click **Test Recorder Launcher**.

The **Test Recorder Launcher** page is displayed:

Figure 1. Test Recorder Launcher



4. Specify values for these fields:
   - **testcaseId**: Name that identifies your test case in the test case XML configuration file.
   - **classname**: Name of the Java class that encapsulates the behavior of you test case.
   - **packagename**: Name of the package in which the Java class resides.
   - **path**: Application server directory in which you want the test case's Java source file to be saved.
   - **configpath**: Application server directory in which you want the test case's configuration file to be saved.
   - **folderPath**: Repository's cabinet and folder path in which to store data (for example, folder into which to import files) that the test case uses. For example:

     `/testcabinet/testdata`
   - **clientDataFolderPath**: Browser machine's directory in which to store data (for example, files to import) that the test case uses.

   Tip: To save time, you can save these field values and load all of them at one time into another test case:
   - To save these field values, change the path, file name, or both, and click **Save State**.

- To load values for these values from a previous state, type the path and file name of the previous state file and click **Load State**.

5. In the **auto login** field, choose whether to automatically or manually log in to a specific repository when test case recording begins:
   - To manually log in, select **No**.
   - To automatically log in, select **Yes** and specify values for these fields:
     - **user**: Name of a user in the repository specified in the **docbase** field.
     - **password**: Password of the user specified in the **user** field.
     - **docbase**: Name of a repository.
     - **language**: Name of a locale.
     - **network location**: An Accelerated Content Services (ACS) network location. This field is only displayed when ACS is enabled for your installation.
6. In one of these fields, specify the URL to the action or component to initially execute when test case recording begins:
   - **URL**: Specify a URL, starting after the Webtop web application root (default is `/webtop`) and including the path to the component or action name. For example, if the web application root is /webtop and the search component path is /webtop/component/search, then specify:

     ```
     /component/search
     ```

     Note: Specifying the URL without the web application root makes it easier to reuse the test case in another web application (which will have a different web application root).
   - **Action/Component Mode**: Select this option and click **Configure**.

     In the recording URL generator page:

     1. **Invoking Categories**: Select **Component** or **Action**.
     2. **For item**: Select the component or action.
     3. **Scope**: Select the scope for the component or action.
     4. **Parameter Contract**: Supply values for the component or action parameters.

        To validate that the correct error dialog box is displayed when a required parameter is not set, do not enter a value or select a variable. (Ignore any warnings,)
     5. **Add a parameter not defined in the action / component**: Add any custom parameters.
     6. Click **OK**.
7. Specify test cases and test suites to execute before (that is, setup) and after (that is, teardown) the execution of your test case:
   - To add a test case or test suite, select either **Setup** or **Teardown**, select a test case or test suite, and click the greater than button (>).

- To order the execution of the test cases or test suites, select a test case or test suite and click the up (^) and down (**V**) buttons.
- To remove a test case or test suite, select the test case or test suite and click the less than button (<).
- To remove all test cases and test suites, click **Reset**.

  Public variables are not removed.

8. To create and delete public variables to be used by all test cases in both setup or teardown for input or output purposes, select **Recorded case will look like this** and click **Edit**, and specify values or other variables for the variables in the **Recorder Test Case Variables** page:
   - To add a variable, enter a variable name in the **Variable** field, a default value in the **Value** field, a short description of the purpose of the variable in the **Description** field, and click **Add**.
   - To delete a variable, click **Delete** in the same row as the variable you want to delete.
9. To create and delete private variables to be used by only a single test case, select the test case, click **Edit**, and specify values or other variables for the variables in the **Test Case Parameters** page:
   a. To use a variable, select the corresponding checkbox in the **Use?** column.
   b. Perform one of these actions:
      - Enter a value in the **Input** column, or
      - Select the corresponding checkbox in the **isVariable?** column and specify a public variable in the **Input** column.

        Tip: To be able to select a public variable to use, you must have defined at least one public variable in Step 8.
10. Click **Record**.

    The **Recorder Monitor** page is displayed.
11. Manually perform the steps you want to record in your test case.
12. To validate specific control values:
    a. Click **Turn On Inspection Mode**.

       Note: In a modal pop-up dialog box, the **Turn on inspection mode** link is displayed in its upper left corner, because you cannot click the **Turn on inspection mode** link in its parent windows.

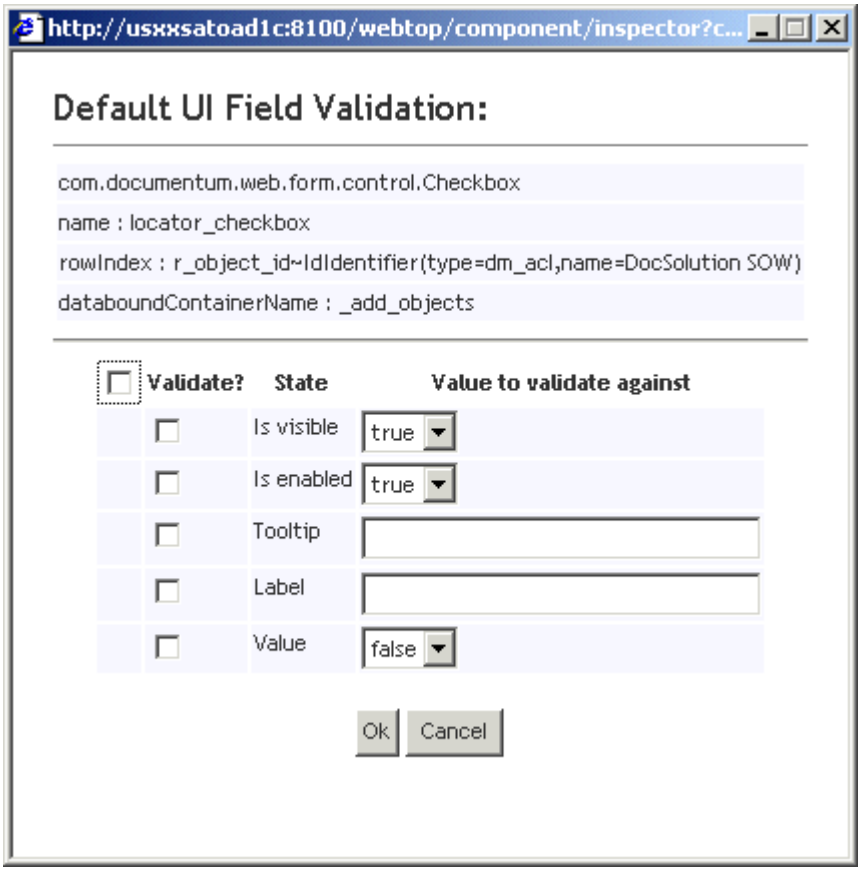
       When you move the pointer over a control, a red box is displayed around the control. Then, you can click the control to display parameters that you can set to validate during test execution.

Figure 2. Inspection mode activated

b. Click a control.

The **Default UI Field Validation** page is displayed.

Figure 3. Default UI Field Validation page



c. Select **Validate?**, select the parameters you want to validate, and select or choose a value to be validated during test execution.

d. If you select a menu control, these additional fields are displayed in which you can validate the state of the MenuGroup, including all its menu items:

- **Menu Item Visible** – Whether a menu item is visible or hidden.
- **Menu Item Enabled** – Whether a menu item is enabled or disabled.
- **Menu Item Label** – Whether a menu item is displayed with a specific value.

e. If you select a tabbar control, you can validate the text of a tab label by specifying the text in the **has tab value of** field.

13. Click **Stop Recording and Run TearDown**.

14. (Optional) Edit, save, and deploy the test case configuration file to Webtop:

a. Click **[edit and save the XML]** (to the right of **Get the XML for the resultant case**).

The XML editing page is displayed.

b. Edit the test case configuration file, change the deployment path in the **Path to deploy the edited xml** field, and click **Save**.

15. (Optional) To run the test case (without modification) on the current Webtop installation, click **[Compile]** (to the right of **Get the Source Code**) to compile the test case Java class.

    Now you can execute your test case. Go to the **WDK Test Framework Tools** page, click **Test Launcher**, click **Refresh ConfigService**, and follow the steps in To execute a test cases and test suites:, page 33.

16. (Optional) Modify the generated test case skeleton configuration file and skeleton class to implement additional validation:
    - Validating object attribute values, page 25
    - Validating actions, page 25
    - Validating whether step is a modal pop-up dialog box, page 26
    - Validating whether a row is selected, page 26
    - Validating XForms control, non-browser, tooltip values, page 26
    -

Validating object attribute values

When an object's attribute values have changed, validation elements are created in the test case skeleton configuration file and corresponding logic is created in the test case skeleton class. The test case validates that the value of an attribute at the beginning of an action is different at the end of the action. You can change these validation elements to compare against specific hard-coded values, variable values, or not-equal comparisons.

For example, in a test case where the title of a document is changed, the test case would fail if the title at the beginning of test case execution was the same as the title at the end.

Validating actions

When an action is executed, validation elements are created in the test case skeleton configuration file and corresponding logic is created in the test case skeleton class. For example, when you complete a checkout action, the document should be locked. If the document is not locked, then validation fails.

Validating whether step is a modal pop-up dialog box

To validate whether the current step is a modal pop-up dialog box, add the ModalPopupValidationEvent() event as an event in the returned events of the step's getClientEvents() method. For example:

```
new ModalPopupValidationEvent();
```

Note: The constructor does not take any arguments.

Validating whether a row is selected

To validate whether a row is selected in a datagrid, add a DatagridRowSelectedValidationEvent instance as an event in the returned events of the getClientEvents() method of the step in which the validation is to occur. The constructor is:

```
DatagridRowSelectedValidationEvent (Component component, String strName,
ArgumentList uniqueArgs, boolean testSelected)
```

Where:

- *component*: name of the component containing the datagrid
- *strName*: name of the datagrid
- *uniqueArgs*: an argument list that identifies the row to validate
- *testSelected*: true, to validate that the row is selected; false, to validate that the row is unselected

Validating XForms control, non-browser, tooltip values

To validate XForms control, non-browser, tooltip values, add the XFormsTooltipValidationEvent() event in the body of the getClientEvents() event of the desired step in which validation is to take place (for example, after a user performed an action that changed the state of the tooltip):

```
new XFormsTooltipValidationEvent(component, name, value)
```

where:

- *component*: name of the component holding the control
- *name*: name of the control

- *value*: the value against which the runtime tooltip value is to be validated

Chapter 4

Making Test Cases Reusable

These topics are included:

Overview

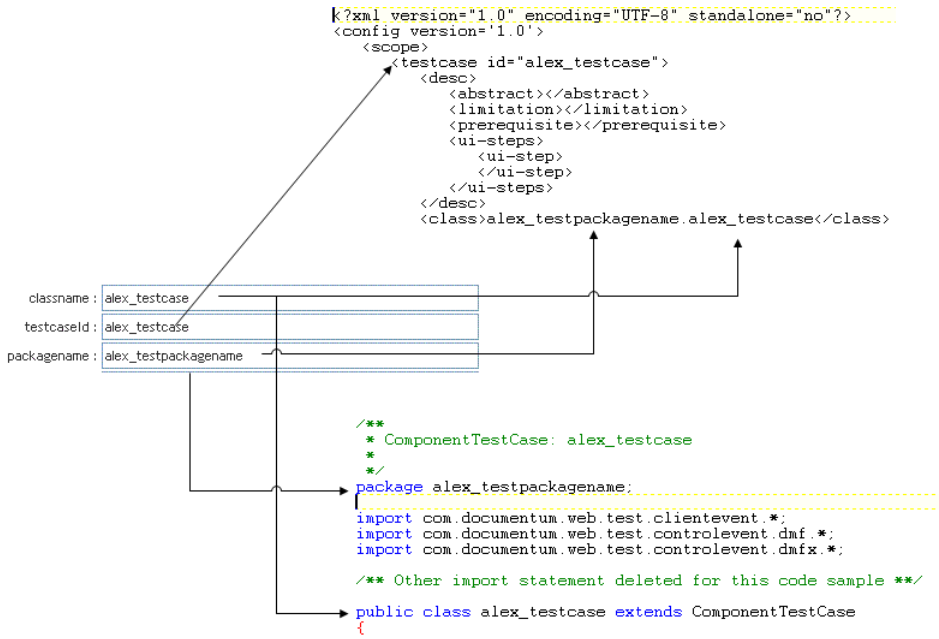
Because recording a test creates a test case with the instance data of your particular test execution, you usually want to make the test case reusable by replacing the instance data with variables to which values can be assigned dynamically at runtime or by the quality assurance engineer before running the test case. For example, when you record a test case that checks out an object, then the object's object ID is recorded in the test case's configuration file skeleton. Consequently, when you run the test case again, if an object with the original object ID does not exist in the repository, then your test case would fail. Requiring the same objects to be present in a repository makes maintenance time-consuming. Instead, you want to replace the hard-coded object ID with a variable to which a value can be assigned dynamically at runtime or by the quality assurance engineer before running the test case.

To replace a test case's instance-specific data with variables, you modify the test case's generated configuration skeleton (and, sometimes, its class skeleton) by specifying public variables, input parameters, and output parameters. Input parameters input values to a test case specified as a teardown test case. Output parameters output values to a test case specified as a setup test case. Public variables are referenced by input and output parameters.

Generated test case configuration and class skeletons

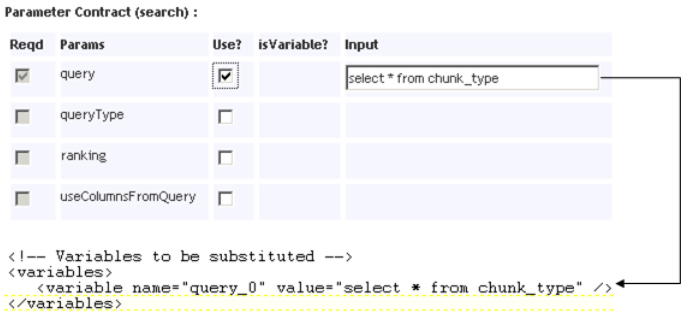
When you record a test case, the class, package, and test case ID values that you specified in the **Test Recorder Launcher** page are used in the test case skeleton class and configuration file. Figure 4, page 30 shows the generated Java class and XML configuration skeleton code that correspond to the **testcaseId**, **classname**, and **packagename** fields in the **Test Recorder Launcher** page.

Figure 4. Test case ID, class name, and package name field values in generated class and configuration skeletons



Other values are also saved to the configuration file. Figure 5, page 30 shows the generated XML configuration skeleton code, the variable element, that corresponds to the **Input** field value of the query parameter.

Figure 5. Input field value in generated configuration skeleton



General procedure to make test cases reusable

To make your test case reusable, you create output parameters that output values from setup test cases and input parameters that input values to teardown test cases.

To make your recorded test case reusable:

1. In the test case configuration file:
   - Create public variables to be referenced in the setup and teardown test cases.
   - For the setup test case, specify output parameters that map to the public variables.
   - For the teardown test case, specify input parameters that map to the public variables.

   If an undefined variable is specified as input to a called test case, an exception is thrown. For example, this exception can occur when a test case calls another setup or teardown test case that specifies an input variable that is not defined in the calling test case.

   For example:

```
<config version='1.0'>
   <scope>
      <testcase id="checkout_testcase">
         <!-- Variables to be substituted -->
         <variables>
          1<variable name="importedObjectId" return="true"/>
            ...
         </variables>
         <setup>
            <list>
               <item>
                  <type>testcase</type>
                  <value>importaction</value>
                  <outputs>
                   2<output name="newObjectId" variable="importedObjectId" />
                     ...
                  </outputs>
               </item>
            </list>
         </setup>
         <teardown>
            <list>
               <item>
                  <inputs>
                   3<input name="useObjectId" variable="importedObjectId"/>
                     ...
                  </inputs>
                  <type>testcase</type>
                  <value>deleteobjectaction</value>
               </item>
            </list>
         </teardown>
      </testcase>
   </scope>
</config>
```

1 For a variable that will be used as an output variable, specify `true` for its return attribute.
2 The variable attribute value matches the name of the public variable.
3 The variable attribute value matches the name of the public variable.

2. In the onComplete method of the setup test case's skeleton class, implement retrieving the value that you want to use and then call the setVariable method to set the variable with that value.

To implement functionality that interacts with a repository, you must use the Documentum Foundation Classes (DFC). See the *Documentum Foundation Classes Development Guide*.

For example, this code sample implements setting the importedObjectId variable (in the test case configuration file) to the object ID of an imported object:

```
public void onComplete(String strAction, boolean bSuccess, Map completionArgs)
      {
         if (m_actionValidator != null)
         {
            m_actionValidator.validateAction(strAction, bSuccess, completionArgs);
         }
         TestObjectInFolder objectInFolder = null;
         if (m_strFolderId != null)
         {
           1objectInFolder = new TestObjectInFolder(getVariable(DOCNAME),
                                                    m_strFolderPath);
         }

         try
         {
            if (objectInFolder.getObjectId() != null)
            {
              2m_strObjectId = objectInFolder.getObjectId();
              3setVariable("importedObjectId", m_strObjectId);
            }
            else
            {
               //fail
               throw new TestCaseFailure("Object is not imported.");
            }

         }
         catch (DfException e)
         {
            throw new TestCaseError("Test Case Failed onComplete() validation method", e);
         }
      }
```

1 In this code example, assume that TestObjectInFolder is a utility class that returns an instance representing an object in the repository.
2 The getObjectId method of the TestObjectInFolder instance is called to return the object ID of the object in the repository
3 The setVariable method sets the importedObjectId variable to the value of the object's object ID (represented by m_strObjectId).

Chapter 5

Executing Test Cases and Test Suites

These topics are included:

Overview

You use the Test Launcher to:

- Execute test cases and test suites
- Construct new test suites

You can substitute values for the default variable values for this test execution only.

General procedure for executing test cases and test suites

To execute a test cases and test suites:

1. Set options in your testframework_config.xml file as described in Configuring the WDK Automated Test Framework, page 12.
2. Launch the **WDK Test Framework Tools** page by specifying this URL:

   `http://host:port/wdk-app/component/testtool`

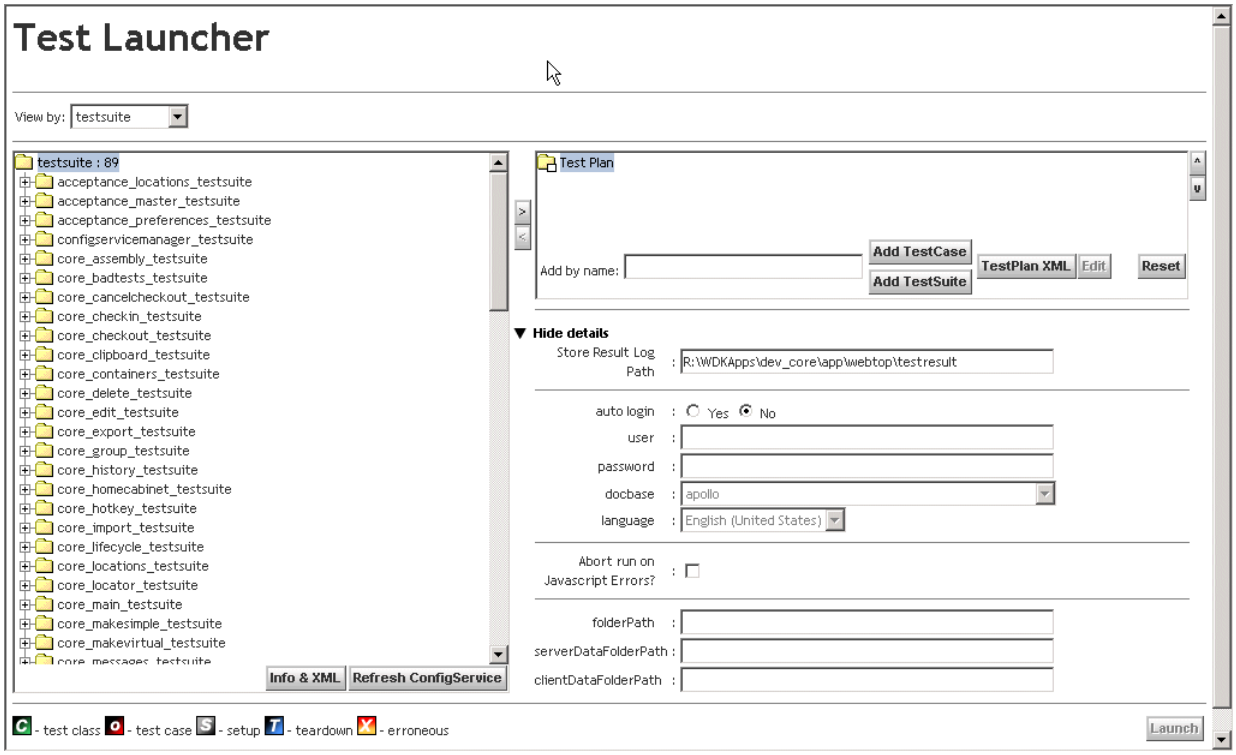
   where:

   - *host*: Name (or IP address) of the machine on which the WDK Automated Test Framework is installed.
   - *port*: Port number at which the WDK Automated Test Framework listens.
   - *wdk-app*: Name of the WDK application that you have installed. Default is `webtop`.

3. Click **Test Launcher**.

   The **Test Launcher** page is displayed:

Figure 6. Test Launcher



4. If there are any new test cases and test suites that you want to display, click **Refresh ConfigService**.
5. In the **View by** field, select **testsuite** or **testcase**.
6. Add the test cases and test suites that you want to execute to the **Test Plan**:
   - To add a test case or test suite:
     - Select a test case or test suite and click the greater than button (>), or
     - In the **Add by name** field, enter the test case or test suite ID and click **Add TestCase** or **Add TestSuite**
   - To order the execution of the test cases or test suites, select a test case or test suite and click the up (^) and down (**V**) buttons.
   - To remove a test case or test suite, click the less than button (<).
   - To remove all test cases and test suites, click **Reset**.

   Tip:

   - **Test Plan** is a temporary test suite that is not saved after you exit this page. If you want to save it, you must save it as a new test suite.

- If you are not sure which test case or test suite to add, select the test case or test suite and click **Info & XML** to view the configuration file.

7. To save all of the test cases and test suites under **Test Plan** as a new test suite:
   a. Click **Test Plan XML**.
   b. Change the id attribute's value in the testsuite element to a usable name.
      For example:
      ```
      <testsuite id="checkout-checkin">
      ```
   c. Change the name and location of the test suite configuration file in the **Path to deploy the edited xml** field and click **Save**.
8. Specify the application server path in which you want to save the test results in the **Store Result Log Path** field.
9. To abort test execution when a JavaScript error occurs and display the JavaScript error in the trace window, check the **Abort run on Javascript Error?** checkbox; otherwise, test execution will continue. You can then troubleshoot and fix the JavaScript error and rerun the test suite.
10. Specify values for these fields:
    - **folderPath**: Repository's cabinet and folder path in which to save test plan data. For example:
      ```
      /testcabinet/unittestdata
      ```
    - **clientDataFolderPath**: Browser machine's directory in which to save test plan data.
11. In the **auto login** field, choose whether to automatically or manually log in to a specific repository when test case recording begins:
    - To manually log in, select **No**.
    - To automatically log in, select **Yes** and specify values for these fields:
      — **user**: Name of a user in the repository specified in the **docbase** field.
      — **password**: Password of the user specified in the **user** field.
      — **docbase**: Name of a repository.
      — **language**: Name of a locale.
      — **network location**: An Accelerated Content Services (ACS) network location. This field is only displayed when ACS is enabled for your installation.
12. To set variable values to use only during this execution of the test suite, select a test case, click **Edit**, and specify values or other variables for the variables in the **Test Case Parameters** page:
    a. To use a variable in this test run, select the corresponding checkbox in the **Use?** column.
    b. Perform one of these actions:
       - Enter a value in the **Input** column, or
       - Select the corresponding checkbox in the **isVariable?** column and specify a variable in the **Input** column.

13. To run all of the test cases and test suites in the test plan, click **Launch**.

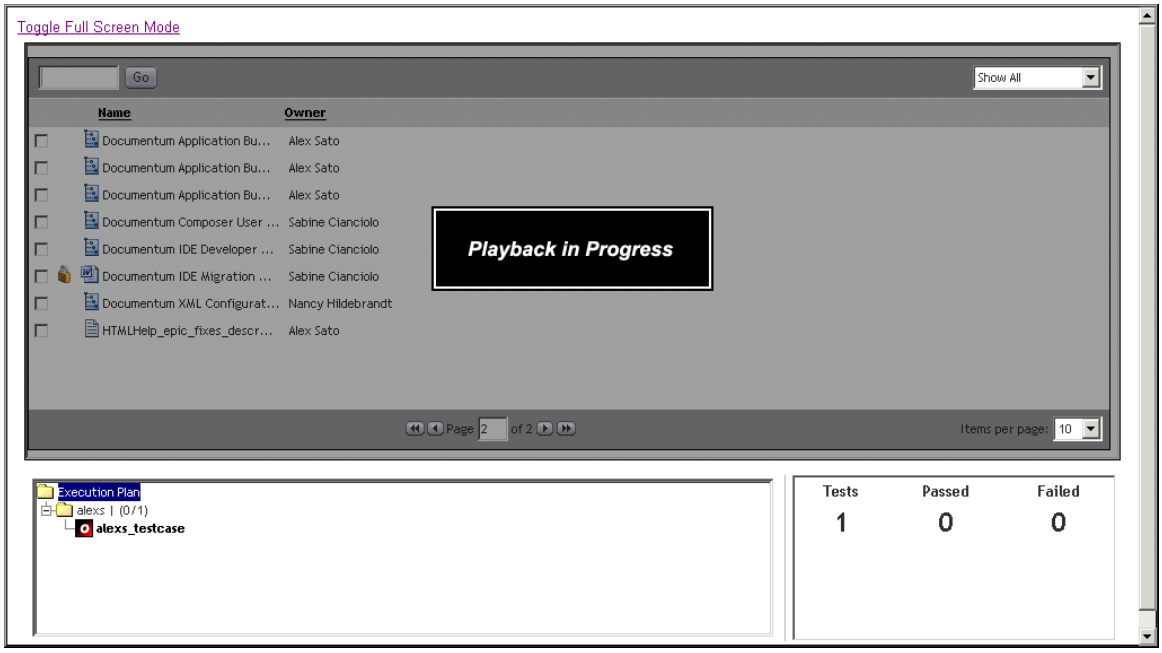
    Note: If you have not specified any test cases or test suites in your test plan, **Launch** is disabled.

    The **Launcher Monitor** page is displayed and the test cases are executed.

    Tip:

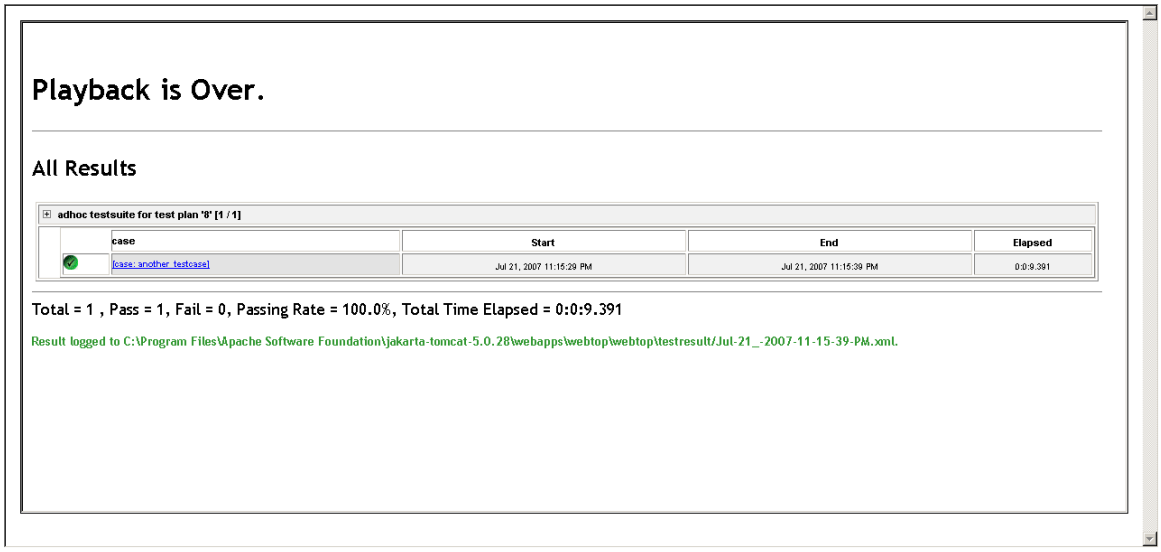
    - Click **Toggle Full Screen Mode**, to expand the playback pane to the full browser window.
    - To hide the **Playback in progress** box and view your application while it runs, move the mouse cursor completely out of the application frame.

Figure 7. Launcher Monitor: playback in progress page



After the test plan is completed, the results page is displayed:

Figure 8. Launcher Monitor: results page



14. Read the test results.

    To understand how to interpret the test results, see Chapter 6, Understanding Test Results.

    a. Click the plus sign (+) next to a test case (under the **All Results** heading) to expand results for the test case.

    b. Click the link in the **case** column to display details about the result (including a stack trace if available).

    Tip: To display the test results as one log file, enter the URL as follows:

    `http://host:port/wdk-app/webtop/testresult_path`

    where:

    - *host*: Name (or IP address) of the machine on which the WDK Automated Test Framework is installed.
    - *port*: Port number at which the WDK Automated Test Framework listens.
    - *wdk-app*: Name of the WDK application that you have installed. Default is `webtop`.

- *testresult_path*: The test results log file's path, starting from the root of the Webtop web application. The test results log file path is displayed at the bottom of the page. For example, if the Webtop web application's root is:

  `C:\Program Files\Apache Software Foundation\jakarta-tomcat-5.0.28\webapps\webtop`

  then, you use the `webtop\testresult/Jul-20_-2007-2-54-17-PM.xml` portion from the test results log file path:

  `C:\Program Files\Apache Software Foundation\jakarta-tomcat-5.0.28\webapps\webtop\webtop\testresult/Jul-20_-2007-2-54-17-PM.xml`

  This results in this URL :

  `http://webtop-machine:8080/webtop/webtop/testresult/Jul-20_-2007-2-54-17-PM.xml`

To view the test results of a previously executed test suite:

1. Launch the **WDK Test Framework Tools** page by specifying this URL:

   `http://host:port/wdk-app/component/testtool`

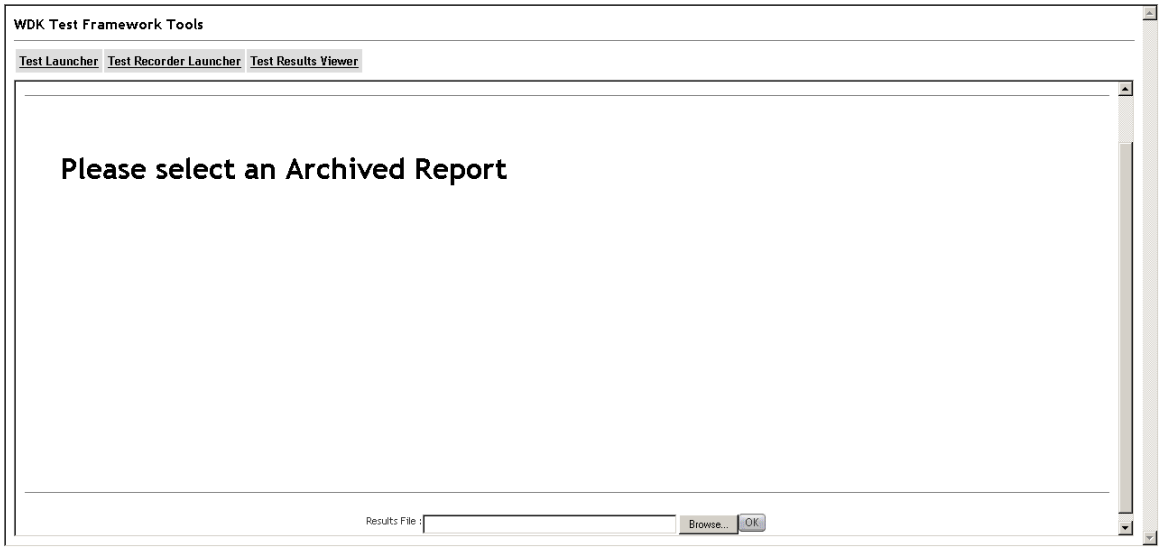
   where:

   - *host*: Name (or IP address) of the machine on which the WDK Automated Test Framework is installed.
   - *port*: Port number at which the WDK Automated Test Framework listens.
   - *wdk-app*: Name of the WDK application that you have installed. Default is `webtop`.

2. Click **Test Results Viewer**.

   The **Test Results Viewer** page is displayed:

Figure 9. Test Results Viewer



3. Click **Browse** and select a file, or enter a path in the **Results File** field, then click **OK**.

Chapter 6

Understanding Test Results

These topics are included:

Overview

Test results consist of these kinds of results:

- Success: The test case succeeded.
- Failure: The test case failed.
- Error: The test case did not complete because of an unexpected error in the test case.

Understanding test failures and errors

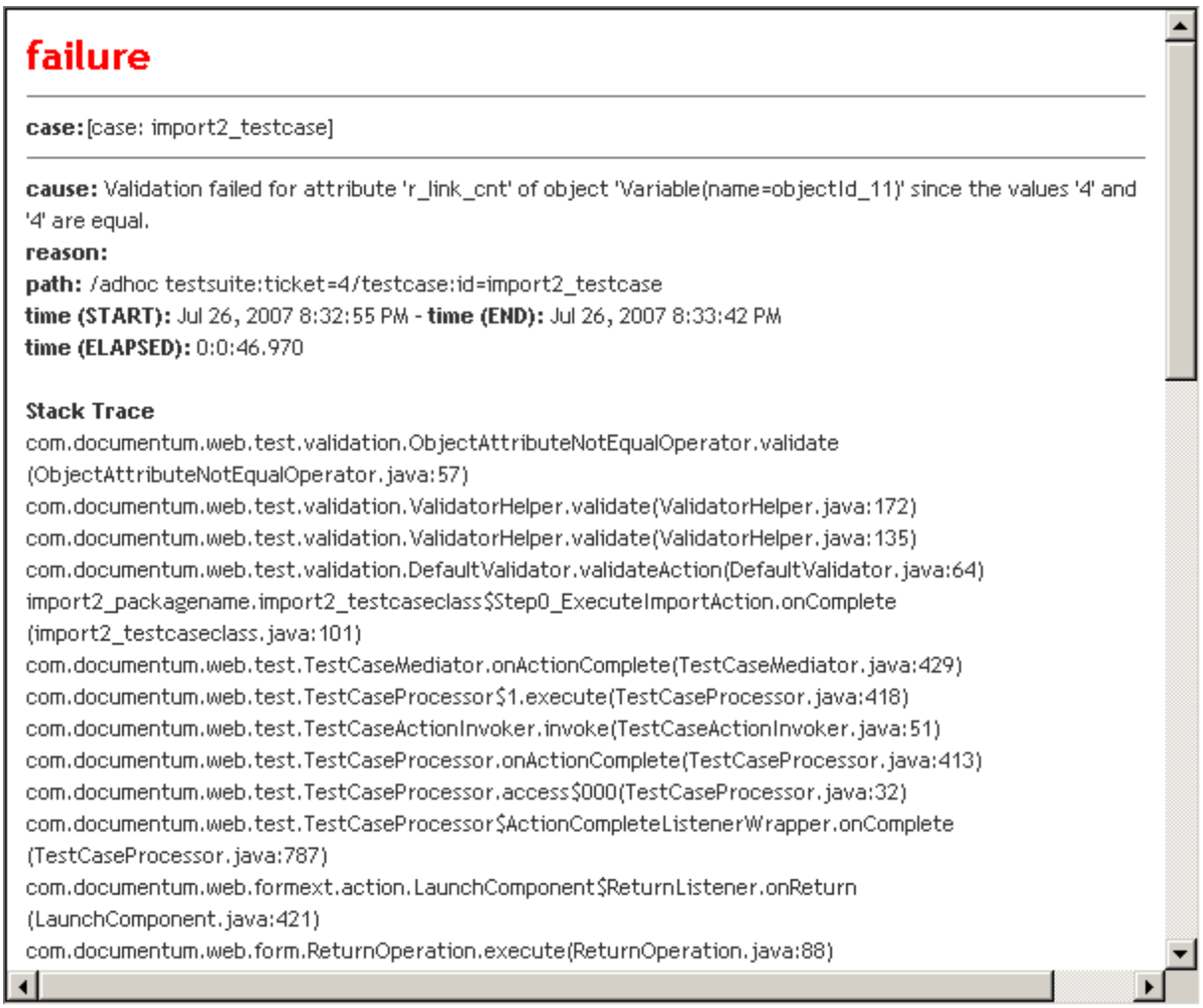
Test errors are errors within the test case's logic or test data, whereas test failures are errors in validation.

In the **All Results** page, clicking a test failure or error link in the **case** column displays a **Failure** (see Figure 10, page 42) or **Error** page that includes these fields:

- **case**: Value of the testcase element's id attribute in the test case configuration file.
- **cause**: Detailed explanation for the test failure or error.
- **reason**: Not used.
- **path**: The test suite (if any) that contains the test case.
- **time (START)**: Date (*Month Day, Year*) and time that the test started.
- **time (END)**: Date (*Month Day, Year*) and time that the test stopped.
- **time (ELAPSED)**: How long the test ran in *Hours*:*Minutes*:*Seconds*:*Milliseconds*.

- **Stack Trace**: The stack trace of the test case execution leading up to the failure.

Figure 10. Failure page

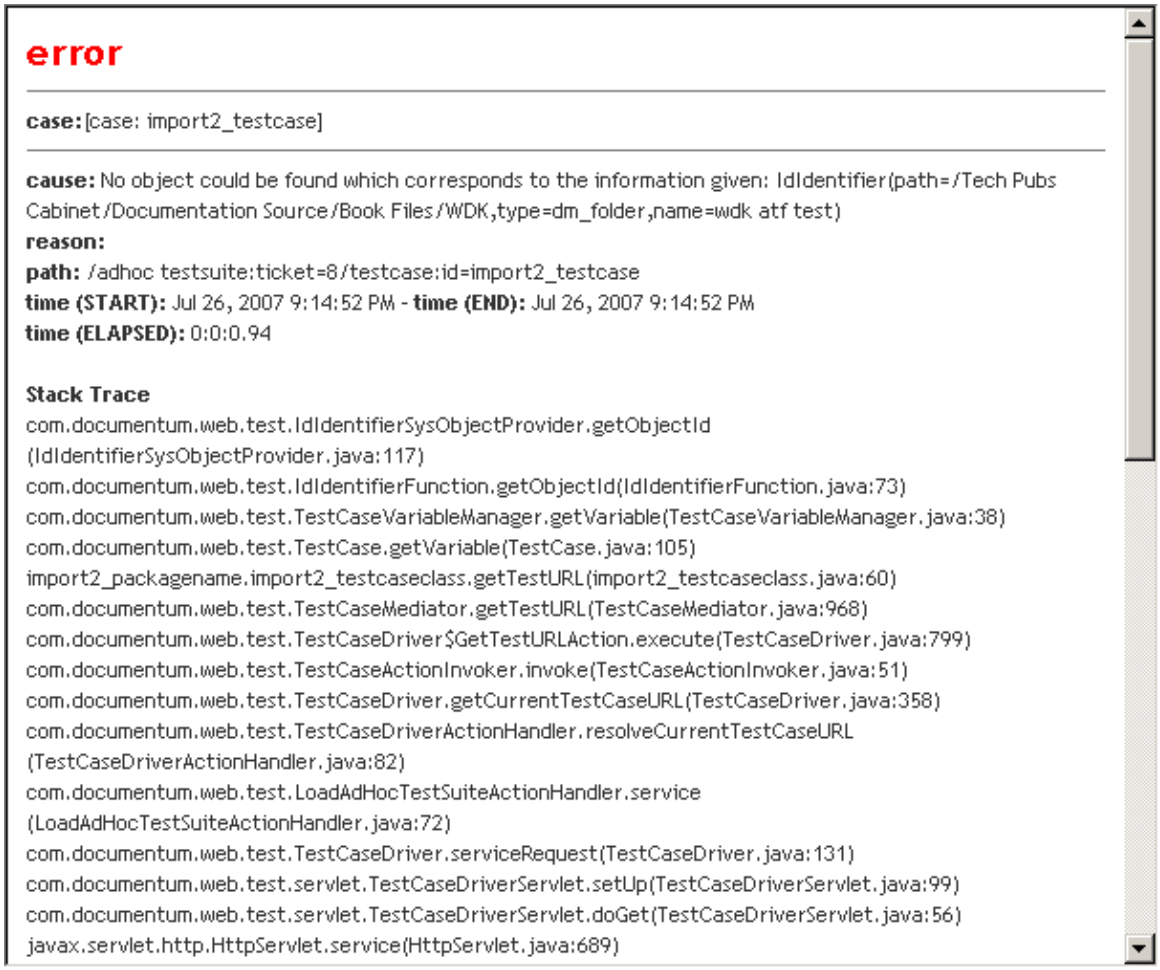


Example 6-1. Troubleshooting a test case error

This example demonstrates fixing a simple test case error.

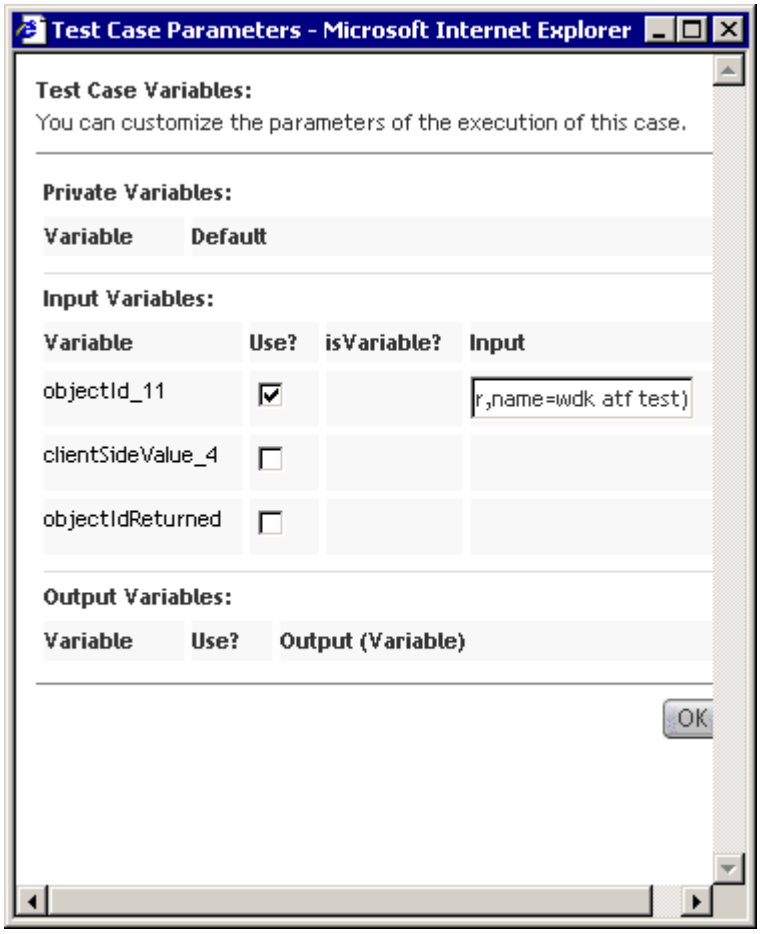
1. The **cause** field indicates that this error was caused by an invalid path or an object that did not exist in the specified location.

Figure 11. Error page: invalid path or non-existent object



2. The invalid path or reference to a non-existent object is a value in a variable, because the stack trace shows a getVariable call.
3. Look in the test case's parameters to find the invalid path or non-existent object reference and correct it.
4. Because only the objectId_11 variable is used and it specifies the same path that was displayed in the error page, the error must be an invalid path. Since the test case already specifies the correct path, it was an error to specify the path for the input variable. To correct the error, uncheck **Use?**.

Figure 12. Test Case Parameters page



Chapter 7

Preparing to Deploy Your WDK Application to a Production Environment

Overview

To improve performance and remove any security issues, remove the WDK Automated Test Framework files and any of your test cases before deploying your custom WDK application to a production environment.

Procedure

1. Follow the procedure for deploying WDK applications in the *Web Development Kit and Webtop Deployment Guide* Version 6.5.
2. Before creating the WAR file, remove the framework by deleting these files and folders from your WDK application installation:

   version.txt
   webcomponent/config/testtool webcomponent/testtool
   webtop/config/testtool
   wdk/config/test/inspector_component.xml
   wdk/src/com/documentum/web/test/controlevent
   wdk/src/com/documentum/web/test/validation/ui/fieldhandler
   wdk/src/com/documentum/web/test/validation/ui/inputassistance
   wdk/src/com/documentum/web/test/validation/ui/inspector
   wdk/src/com/documentum/web/test/recorder/DatagridRowDoubleClickUserEvent.java
   wdk/src/com/documentum/web/test/recorder/DatagridRowMouseDownUserEvent.java
   wdk/src/com/documentum/web/test/recorder/TreeNodeUserEvent.java
   wdk/src/com/documentum/web/test/TreeNodeIdentifierFunctionProvider.java
   webtop/src/com/documentum/webtop/test/
   WebtopBrowserTreeNodeIdentifierFunctionProvider.java
   WEB-INF/classes/com/documentum/webtop/testcase/samples
   webtop/src/com/documentum/webtop/testcase/samples

Tip: You can use this ANT snippet to create a WAR file without the files installed from the WDK_TestFramework_2.5.zip file:

```
<target name="create.war" description="Creates webtop.war file from basedir of app/">
  <!-- Defines base.dir to be app/, please change if the root of your webapp is
       at a different location -->
  <property name="base.dir" value="app"/>

  <!-- Deletes the old webtop.war file from ${base.dir}/ if present,
       please backup your copy -->
  <delete file="${base.dir}/webtop.war" quiet="true" />

  <!-- Creates war file to ${base.dir}/webtop.war, excluding test framework files -->
  <jar destfile="${base.dir}/webtop.war"
       compress="false">

    <fileset dir="${base.dir}">
    <!-- Excluding testtool config files -->
    <exclude name="version.txt"/>
    <exclude name="webcomponent/config/testtool/**"/>
    <exclude name="webcomponent/testtool/**"/>
    <exclude name="webtop/config/testtool/**"/>
    <exclude name="wdk/config/test/inspector_component.xml"/>
    <exclude name="wdk/src/com/documentum/web/test/controlevent/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/controlevent/dmf/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/controlevent/dmf/databound/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/controlevent/dmfx/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/controlevent/dmfx/xforms/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/validation/ui/fieldhandler/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/validation/ui/inputassistance/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/validation/ui/inspector/*.*"/>
    <exclude name="wdk/src/com/documentum/web/test/recorder/
                   DatagridRowDoubleClickUserEvent.java"/>
    <exclude name="wdk/src/com/documentum/web/test/recorder/
                   DatagridRowMouseDownUserEvent.java"/>
    <exclude name="wdk/src/com/documentum/web/test/recorder/TreeNodeUserEvent.java"/>
    <exclude name="wdk/src/com/documentum/web/test/
                   TreeNodeIdentifierFunctionProvider.java"/>
    <exclude name="webtop/src/com/documentum/webtop/test/
                   WebtopBrowserTreeNodeIdentifierFunctionProvider.java"/>
    <!-- Excluding samples files -->
    <exclude name="WEB-INF/classes/com/documentum/webtop/testcase/samples/**"/>
    <exclude name="webtop/src/com/documentum/webtop/testcase/samples/**"/>
    </fileset>
  </jar>
 </target>
```

Chapter 8

Troubleshooting

These topics are included:

Debugging test cases

To turn on tracing for the test recorder, set the com.documentum.web.test.Trace.TESTRECORDER flag to true. This flag traces the recording and production of a test case.

Known issues during recording

Before recording a test case, you should make sure the manual testing of the test scenario is working properly. Assuming a clean environment (an environment in which everything is prepared for recording or playback, repeatedly), any other errors are most likely the test framework bug (or lack of support for the feature being recorded).

Known issues during playback

Errors during playback could be caused by one of the following:

- Lack of repository objects

  The section Lack of repository objects, page 48 contains more details.
- Presence of repository objects

  The section Presence of repository objects, page 48 contains more details.

- Preference mismatch

  The section Preference mismatch, page 48 contains more details.
- Hard-coded object IDs and names

  The section Hard-coded object IDs and names, page 49 contains more details.
- Framework bug

  The section Framework bug, page 49 contains more details.

Lack of repository objects

The test tries to act on an object that does not exist. Tests must ensure that the environment is set properly before the test execution. In such cases, the manual testing reveals that the object or control being acted upon does not exist.

Symptoms

Symptoms during playback are:

- Test case times out
- A JavaScript error occurs mentioning inability to find the control

Presence of repository objects

The test tries to create an object that already exists. Tests must ensure that the environment is set properly before the test execution. In such cases, the manual testing reveals that an object, that already exists, is being created.

Symptoms

Symptom during playback is:

- Test case times out

Preference mismatch

The test does not initialize its preference and fails as a result. For an instance during recording, the number of items per page may have been manually set to 50, but during playback it could default

to 10. This could cause a document that was expected to be on page one, to be on another page and hence cause failure.

Symptoms

Symptoms during playback are:

- Test case times out
- A JavaScript error occurs mentioning inability to find the control

Hard-coded object IDs and names

The test may have hard-coded values in test XML file. For an instance, a test that inspects a repository name will have that repository name hard coded in the XML file. When you run this test against a different repository fails.

Any hard-coded values such as object ids and names should be properly accounted for and parameterized.

Symptoms

Symptoms during playback are:

- Test case times out
- A JavaScript error occurs mentioning inability to find the control

Framework bug

If any of the known issues such as lack of repository objects, presence of repository objects, preference mismatch and hard-coded object ids and names do not occur, there could be many more other possibilities and here are a few:

1. Playback fails immediately after recording (on the same version that test was recorded and in clean environment). The possibilities could be:
    - Sufficient time was not given for a page to render during recording.
    - There exists a benign JavaScript error in the product that causes failure during playback due to more stringent JavaScript requirements by the framework. Though this manifests as a test case failure, it is actually a product bug.
    - Bug in the framework

2. Playback worked on the version recorded, but fails on newer version of WDK/Webtop/Framework. The possibilities could be:
   - Feature has changed by design - Re-recording will be needed if a feature has changed by design. It is also possible to modify the test case rather than re-recording. Though this manifests as a test case failure, it is an intentional change which requires rework.
   - Feature has regressed
   - Framework has regressed - To validate if this is a framework bug; the manual testing of the test scenario should work fine.

Symptoms

Symptoms could vary, but here a few:

- Test case times out
- A JavaScript error occurs mentioning inability to find the control

Known issues during result view

If the result page does not show the correct data, you have to make sure that the recording completes all the actions.

WDK trace flags to trace Test Harness at runtime

Two WDK trace flags are defined to trace the Test Harness at runtime:

- com.documentum.web.test.Trace.TESTCASE: This flag traces the traversal of test suites and test cases and the execution of test cases.
- com.documentum.web.test.Trace.TESTSTEP: This flag traces the traversal of test steps and the execution of test steps.

- com.documentum.web.test.Trace.TESTRECORDER: This flag traces recorder preferences, listeners, and other recorder actions.

Chapter 9

Implementing unit tests for components

These topic are included:

Overview

Component unit testing is testing components without having to perform any manual actions. You can implement unit tests for components by extending ComponentUnitTestCase and providing, in the component's test case configuration file, a mapping of variables to names of component test case class methods to run. You execute the unit test by running its test case from within the WDK Automated Test Framework. You can also override getTestMethod() to return a string array of the names of the methods in the component test case class to run.

General procedure

To implement a class and configuration file for unit testing:

1. Extend ComponentUnitTestCase and implement methods that test the component's functionality.
2. Create a test case configuration file that specifies the class and its methods.

   You specify these elements in the test case configuration file:

```
<config version='1.0'>
  1 <scope>
     2 <testcase id="wtapplevelmodification">
       3 <class/>
       4 <variables>
          5 <variable name="testMethod_number" value="method_name" />
         ...
         </variables>
      </testcase>
   </scope>
```

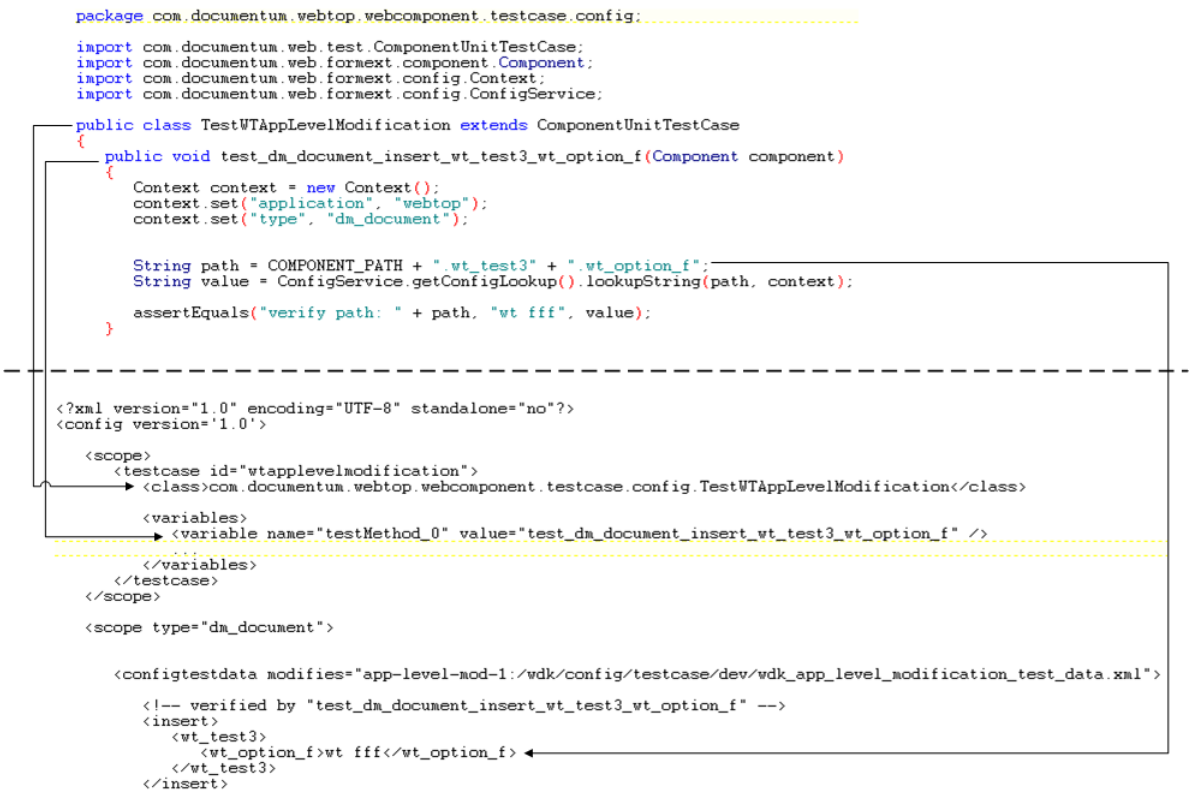
```
6 <scope type="dm_document">
   7 <configtestdata modifies="app-level-mod-1:/wdk/config/testcase/dev/wdk_
                               app_level_modification_test_data.xml">
       8 <insert>
           <wt_test3>
              <wt_option_f>wt fff</wt_option_f>
           </wt_test3>
        </insert>

        <insertafter path="wcl_test2">
           <wt_option_a>wt aaa</wt_option_a>
        </insertafter>

        <replace path="test.wcl_option_c">
           <wcl_option_c>wt replace wcl_option_c</wcl_option_c>
        </replace>
        ...
     </configtestdata>
  </scope>
  ...
</config>
```

1 Container for the entire component unit test configuration.
2 Contains information required for the WDK Automated Test Framework to execute the class.
3 Name of the package and class.
4 Container for <variable> elements.
5 Specifies a method in the class to execute. The WDK Automated Test Framework executes the class method specified in the value attribute, whenever the name attribute value is in the format: `testMethod_`*`number`*, where *number* is an integer; each value attribute's *number* value must be incremented by 1 from the previous <variable> element's value attribute. (The <variable> elements do not have to be ordered sequentially.)
6 Specifies the context for the contained test data using standard WDK scoping.
7 Container for test data. The modifies attribute specifies any other configuration file's test data.
8 (Optional) Specifies test configuration data. Your test case class logic must be able to parse these XML structures.

Example 9-1. Verifying path to component

This unit test verifies the path to a component. Figure 13, page 55 shows the elements in the unit test configuration file that correspond to the code in the class.

Figure 13. Class name, method name, and variables in class and configuration files



The test_dm_document_insert_wt_test3_wt_option_f method verifies the path to the component. The statement, `String path = COMPONENT_PATH + ".wt_test3" + ".wt_option_f"`, sets the path to the component using variables (wt_test3 and wt_option_f) from the test case configuration file. The statement, `assertEquals("verify path: " + path, "wt fff", value)`, asserts that the path variable value is equal to the value: `wt fff`.

Chapter 10

Modifying Existing Test Cases to Use New Features

These topics are included:

Overview

These topics describe how to directly modify your existing test cases to use new features without having to re-record them.

WDK ATF 2.5

These features were introduced in version 2.5.

Changing an existing IdIdentifier to reference a specific object version

To change an existing IdIdentifier to reference a specific object version, add a version clause to the IdIdentifier using this syntax:

```
IdIdentifier(path=folder_path,type=object_type,name=object_
name,version=version_number_label)
```

where:

- *folder_path*: Path to the folder in which the object resides.

- *object_type*: Object type of the object.
- *object_name*: Object name of the object.
- *version_number_label*: Version number (for example, 1.1) or symbolic version label (for example, DRAFT).

For example, if the following IdIdentifier is in your existing test case configuration file:

```
IdIdentifier(path=SomeFolderPath,type=dm_document,name=SomeDoc)
```

Then, to change it to reference version 1.1 of the SomeDoc document, modify it as follows:

```
IdIdentifier(path=/SomeFolderPath,type=dm_document,name=SomeDoc,version=1.1)
```

Changing an existing IdIdentifier to specify that an attribute references an object

To specify that an attribute references an object, use this this syntax:

```
IdIdentifier(name=object_name, attachment=IdIdentifier(...))
```

where:

- *object_name*: Object name of the object.

You use the IdIdentifier syntax in the attachment parameter to specify the object that the attribute references.

Appendix A

Unsupported features

The WDK Automated Test Framework does not support testing these features:

- Virtual Link
- Browser's File Chooser interaction (for example, HTTP content transfer)
- Drag and Drop
- Safari on the Macintosh OS
- Test cases that run multiple browser windows

Appendix B

New Features, Usability Improvements, and Fixed Bugs

See the *WDK Automated Test Framework Development Guide* for a list of new features, usability improvements, and fixed bugs that apply to extending the WDK Automated Test Framework.

Version 2.5

Test Recording

- If you select a menu control, you can validate the state of the MenuGroup, including all its menu items. (New feature)
- You can specify a locale and Accelerated Content Services (ACS) network location when using the autologin feature. (New feature)
- To validate that the correct error dialog box is displayed when a required parameter is not set, do not enter a value or select a variable when configuring action or component parameters. (New feature)
- During test recording, an inspection link (**Turn on inspection mode**) is displayed in the upper left corner of the modal pop-up dialog box , because you cannot click the inspection link in its parent window. You can also validate whether a step is a modal pop-up dialog box. (New feature)
- If you select a tabbar control, you can validate the text of a tab label by specifying the text in the **has tab value of** field. (New feature)
- You can record and playback user actions on the AttachmentSingle, XFormsDateInput, XFormsDateTime and XFormsSliderControl controls. (New feature)
- You can validate XForms control, non-browser, tooltip values by using the XFormsTooltipValidationEvent() event. (New feature)

- To use the value instead of locale-specific labels, set <localeagnosticrecording> to true. (This element is in <uivalidation> in testframework_config.xml in the WDK layer.)

  During test execution, producers for DropDownListEvents and ListBoxEvents use the value instead of writing out locale-specific labels as choices to select.

  You should set this option to true when you want to ignore locale-specific labels. (New feature)
- You can now record test cases in which Shift-right arrow (->) is used to select multiple rows within a datagrid. (New feature)
- Some controls can only be identified by indexes; for example, an input data table with text fields in its rows that are logically identified as first row, second row, and so forth. The SetValueEvent, ClickEvent, DataPagingEvent events take an index parameter, so the events can operate on the indexed control of the specified name. (New feature)
- Records test cases where the last user interaction does not result in a page refresh. For example, when recording is stopped after the user selects something in the doclist grid and no page refresh occurs. In these cases, a LastStepClientEvent is recorded as the last client event; the LastStepClientEvent indicates the end of the current test case and that execution is to proceed to the next test case. (New feature)
- By default, these new valid arguments are available (New feature):
  - comp_id
  - accessorName
  - queue_name
  - appname
  - object_name
  - localeId
  - objectName
- TreeNodeIdentifiers are supported for com.documentum.web.formext.control.docbase.DocbaseFolderTree. (New feature)
- These objects are supported for IdIdentifier (New feature):
  - dmc_workqueue
  - dmc_workqueue_doc_profile
  - dm_client_registration
  - dmi_queue_item

  Note: This is a best effort identification that matches the newest object with the recorded attributes.
- An IdIdentifier can now reference a specific object version, instead of only the current version. (New feature)
- An IdIdentifier can now reference an attribute that is an object. (New feature)

- The test recorder no longer throws a null pointer exception when inspecting a Datagrid that has columns without labels. (Bug fix)
- Fixed text filters not working correctly while recording. (Bug fix)
- Fixed recording dropdown list values with non-XML safe characters in the label. (Bug fix)
- Fixed recording prompted text, non-user triggered events. (Bug fix)
- Fixed recording UnaryValueDataColumnFilterControl, non-user triggered events. (Bug fix)
- The test recorder now correctly records Mouseover and Clicks on contextmenu items when more than one type of object is in the grid. (Bug fix)
- Fixed refresh on frameset forms not writing out a step. (Bug fix)
- Fixed inspection span from distorting the DOM rendering. Keep inspection span boxes from affecting the existing rendering of the DOM by overlaying the box instead of changing the display style of the span box. (Bug fix)
- Fixed resolving of action menu items failing by also relying on the action instead of just the name of the control (since there may be multiple action menu items with the same name that fire different actions). (Bug fix)
- The Show/Hide detail link in testlauncher/testrecorderlauncher no longer loses its state beyond refresh. (Bug fix)
- SetValueEvent now fires onvaluechange on the client side when autocomplete is off. (Bug fix)
- Test framework no longer breaks the product when Docbase attributes have international characters. (Fix for bug 150800)
- Test recorder no longer produces non-playable test cases due to recording steps on non-component forms. (Bug fix)
- setVariables now work in the teardown method. (Bug fix)
- Fixed the test recorder from recording events fired due to inline request responses, since they will be fired by themselves. In other words, the framework will only record the event that fires the inline request and not any subsequent events that are fired because of the response of such requests. (Bug fix)

Test Execution

- If you have not specified any test cases or test suites in your test plan, **Launch** is disabled. (Usability improvement)
- You can hide the **Playback in progress** box to keep it from obstructing your view of the application while debugging. To hide the **Playback in progress** box and view your application while it runs, move the mouse cursor completely out of the application frame. (Usability improvement)
- If an undefined variable is specified as input to a called test case, an exception is thrown. For example, this exception can occur when a test case calls another setup or teardown test case that specifies an input variable that is not defined in the calling test case. (New feature)

- To abort test execution when a JavaScript error occurs and display the JavaScript error in the trace window, check the **Abort run on Javascript Error?** checkbox; otherwise, test execution will continue. You can then troubleshoot and fix the JavaScript error and rerun the test suite. (New feature)
- When selecting test cases and test suites to execute, you can now select them by typing in test case and test suite IDs, instead of browsing and selecting them. (Usability improvement)
- You can set these test execution options:
  - To continue executing test cases even after a user interface validation failure and report them after test case execution completes, set <completetestonfailure> to true. (This element is in <uivalidation> in testframework_config.xml in the WDK layer.)

    You should set this option to true when you do not want to terminate test execution after encountering the first user interface validation failure. (New feature)
  - To set action menu item click events to be delayed, specify the desired number of milliseconds in <menueventdelay>. (This element is in <uivalidation> in testframework_config.xml in the WDK layer.)

    You should set this option when the menu takes a long time to render, because if an ActionMenuItem's click JavaScript event fires immediately after clicking a menu, the menu item might not be found. (New feature)

Index

R

S

T

U

V