



**Adobe Flex™ 2**

Testing Flex Applications with Mercury QuickTest Professional



© 2007 Adobe Systems Incorporated. All rights reserved.

Testing Flex™ Applications with Mercury QuickTest Professional™

If this guide is distributed with software that includes an end-user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end-user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Flex, Flex Builder and Flash Player are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Linux is a registered trademark of Linus Torvalds. Solaris is a registered trademark or trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners. Quicktest and Mercury Quicktest Professional are registered trademarks or trademarks of Mercury Interactive Corporation or its wholly-owned subsidiaries, Freshwater Software, Inc. and Mercury Interactive (Israel) Ltd. in the United States and/or other countries.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Macromedia Flash 8 video is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>. This product includes software developed by the OpenSymphony Group (<http://www.opensymphony.com/>). Portions licensed from Nellymoser ([www.nellymoser.com](http://www.nellymoser.com)). Portions utilize Microsoft Windows Media Technologies. Copyright (c) 1999-2002 Microsoft Corporation. All Rights Reserved. Includes DVD creation technology used under license from Sonic Solutions. Copyright 1996-2005 Sonic Solutions. All Rights Reserved. This Product includes code licensed from RSA Data Security. Portions copyright Right Hemisphere, Inc. This product includes software developed by the OpenSymphony Group (<http://www.opensymphony.com/>).

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Notice to U.S. government end users. The software and documentation are “Commercial Items,” as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Contents

<b>Chapter 1: Working with QuickTest Professional</b> .....	<b>5</b>
Introduction to the testing process .....	5
Deploying Flex files .....	6
About the Flex installed testing files .....	7
About the testing environment .....	7
Test creation overview .....	8
About the QTP object model .....	8
Storing objects .....	9
About the application hierarchy .....	11
Recording tests .....	12
About the testing script .....	13
Identifying objects .....	14
About operations .....	15
Using checkpoints .....	17
Standard checkpoints .....	17
Bitmap checkpoints .....	18
Table checkpoints .....	18
Using common methods and properties .....	18
Playing back Flex tests .....	19
<b>Chapter 2: Advanced Concepts</b> .....	<b>21</b>
Working with containers .....	21
Working with Repeater objects .....	22
Working with data-driven and List-based controls .....	22
Troubleshooting .....	22
General troubleshooting .....	22
Logging .....	23
Resolving playback synchronization issues .....	24
Delaying startup test time .....	26
Making a test fail but continue to the end .....	28
Limitations of automated testing with Flex .....	29



# Working with QuickTest Professional

This topic describes how to work with Mercury QuickTest Professional™ and Adobe Flex applications.

## Contents

Introduction to the testing process .....	5
Test creation overview .....	8
Recording tests .....	12
About the testing script .....	13
About operations .....	15
Using checkpoints .....	17
Using common methods and properties .....	18
Playing back Flex tests .....	19

## Introduction to the testing process

Large, complex applications must be tested to make sure that user interactions do not create run-time exceptions and that connections to data services are functioning properly.

This document assumes that there are two roles in the testing process: the QA testing professional and the Flex developer. It assumes that the QA testing professional does not know how to develop Flex applications and does not have access to the Flex source code, the Flex compiler, or the Flex documentation. The Flex developer, on the other hand, does not necessarily know how to use a testing tool, such as Mercury QuickTest Professional (QTP).

Flex documentation for developers includes a set of guidelines for creating testable applications. These guidelines include instructions on how to compile testable applications, how to instrument custom components so that they can be tested, and how to write applications so that the testing process works more smoothly. For information about the guidelines, see the *Flex 2 Developer's Guide*.

Before you can test a Flex application, a Flex developer compiles a SWF file for you. The developer also provides you with an HTML file that embeds the SWF file in it. This is known as the HTML wrapper. You deploy these two files to any web server.

Some Flex applications require the use of external resources, such as web services or remote objects, so you should ensure that your web server can access these resources. You might be required to add a permissions file to the target server to allow your Flex application to access the server. For more information, ask the Flex developer.

To run a test, you launch QTP and request the HTML wrapper. QTP records the interaction with the Flex application as it would with any other target type. You should then be able to rerun your tests against the Flex application as long as the web server remains accessible.

## Deploying Flex files

Before you can test the Flex application, you must deploy the application and its supporting files to a location that is accessible by QTP. These files include:

- Application SWF file
- HTML wrapper and wrapper files (the wrapper files can include SWF files, JavaScript files, and other files)
- RSL SWC files
- Module SWF files
- Helper files such as theme SWC files, style SWF files, and image, video, and sound files

You deploy the Flex application and HTML wrapper file to any location on your HTTP server. You must be able to access this location from QTP.

In some cases, files used by the Flex application might be located on a remote server. In this case, you must be sure that the files are accessible through any firewalls. Furthermore, the remote servers might need to include cross-domain permission files to allow remote access. For more information, you should discuss this with the Flex developer.

If necessary, you can password-protect the HTML wrapper and use the QTP authentication mechanism to access it. For more information, see the QTP documentation.

## About the Flex installed testing files

The Flex support for QTP requires that you install two ActiveX plug-ins on the testing machine. These plug-ins provide the necessary communication layer between Flex and QTP. One plug-in runs inside of QTP, and the other runs inside the browser. The browser plug-in is signed, and is designed to run in Microsoft Internet Explorer 6 on Windows only. Together, these plug-ins ensure that Flash Player and the Flex framework can send the appropriate script interactions to QTP during record mode.

Similarly, the plug-ins provide the mechanism for QTP to send script commands to the Flash Player and Flex during playback. The browser plug-in uses the Flash Player `flash.external.ExternalInterface` class and, therefore, requires the correct security context for `flash.external.ExternalInterface` to run. If the SWF file runs from a web server, the Flash Player automatically enables `flash.external.ExternalInterface`. If the SWF file runs from a local file system, the SWF file must be trusted for `flash.external.ExternalInterface` to be enabled and for the browser plug-in to work properly. If the plug-ins have problems loading, the cause is written to the Flash Player debug log file.

## About the testing environment

Flex component names reflect the nature of the component. For example, a button is a `Button` control in the Flex programming environment. In the testing environment, the name of each testable control is prefixed with `Flex`; for example, a `Button` control is known as `FlexButton` in the QTP scripts. This prevents controls from different types of applications from being confused in your scripts.

All visual Flex components can be scripted in your quality control tests. Most of their properties and methods can be recorded and then played back. Each method stores a certain number of properties.

When you create a test for the first time, notice that visual controls are usually referenced inside containers. Not all containers that are used by Flex programmers are reflected in the testing environment. For example, a set of buttons might be inside a horizontal box container (or `HBox`), but that `HBox` control might not show up in the test script. Restricting the controls in the scripts makes the scripts easier to read and keeps the hierarchies as short as possible without losing detail.

The `tea.html` document includes a complete list of Flex components that you can test. You can also view the testable methods and properties of these components in that document.

# Test creation overview

When you record a test, QTP records the lines of the test script correlating to each action that you performed. Each line typically represents an action that you carried out on a component that appears on the screen. Those components are called objects and the actions you perform on them (such as clicking a button) are called operations.

The basic building block of a test is the test object. Objects are stored in the QTP object repository. You should understand how QTP identifies objects and adds them to the object repository. For more information, see “Identifying objects” on page 14.

To create a narrative for a test, you record a series of operations or events on the test objects. It is important to understand which events are recognized by QTP. For Flex developers who want to add events to the list of supported ones for each object, see the *Flex 2 Developer’s Guide*.

Most QTP tests use checkpoints to compare output values against known values. Flex supports a subset of the types of checkpoints that are used in QTP. For more information, see “Using checkpoints” on page 17.

If you encounter problems while testing Flex applications, you should refer to “Troubleshooting” on page 22. This section describes error codes and their meanings, and describes some common problems that can occur when using the Flex plugin with QTP.

## About the QTP object model

Before you create a test, it is important to understand how the object model of the QTP and Flex are integrated. The *test object model* is a set of object types that QTP uses to represent objects that are used in your application. The test object model maps the QTP test objects to the Flex objects in your application. For example, a Button control in a Flex application is recognized as a FlexButton in the QTP object model.

The QTP test objects store properties and record events during the test. Each object has its own set of properties and methods that QTP can record. Test objects do not support all events and properties of the Flex objects that they correspond to, but they do support the events that are most commonly associated with users’ gestures.

The *object repository* is a list of all objects used in the test. To add an object to a test, you must add it to the object repository. To remove an object from the test, you remove it from the object repository.

*Operations* are actions that you perform on objects. They are equivalent to a Flex event, but are generally at a higher level. For example, when you click on a Button, QTP records a “click” operation. QTP does not record the mouseDown and mouseUp events, which essentially make up the click event.

Each operation has one or more arguments that define information about the event. This information includes details such as what keys were held down when the operation occurred or the index of a selected item in a list.

Not all controls have operations. For example, the FlexRule controls do not have operations. You can still add these controls to the object repository and use checkpoints to test object properties. Other controls without operations, such as FlexRepeater and FlexDisplayObject, are low-level objects that you generally do not have to interact with for your tests.

Automated testing requires that each component be identifiable by a set of persistent unchanging properties, with one main one, automationName, which is human readable and easily associated with its on-screen counterpart. Collectively, this set of properties is called the automation ID. QTP stores the automation ID in a repository. A change to a component’s ID results in multiple versions of the object in the repository. Additionally, QA engineers can create data driven scripts by changing these IDs with IDs from a database.

QTP specifically avoids treating list items as components, because the identifiers for list items can change in an editable list, and identifiers can repeat (for example, a data grid with rows that have repeating text). Also, creating and or requiring entries in the repository for list items can create a huge burden for data driven scripts, because you must put all of your list items in the repository and change many objects in the repository.

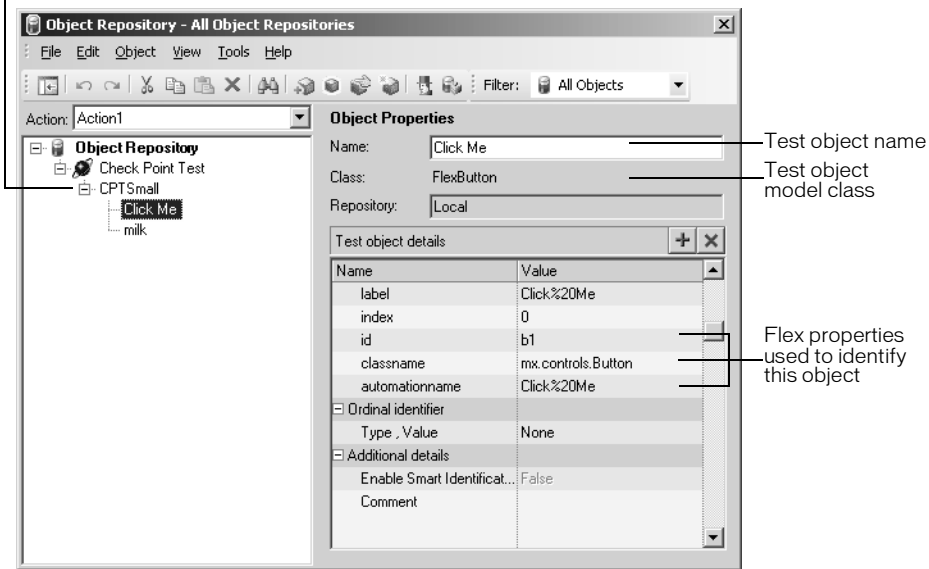
## Storing objects

With QTP, you can record and play back your interaction with the Flex application or you can write the script yourself. It is generally easier to have QTP record your actions with a Flex application, and then modify that recorded script. During a recording session, QTP adds the Flex objects that you interact with to the object repository. Otherwise, you would have to define each object separately before using it in your own script.

When QTP encounters a Flex control during a test, it stores a set of properties that describe that object and stores a reference to it in the object repository. The definition of the Flex control in the QTP object repository contains enough information for QTP to map that Flex control (such as, Button) to a test object (such as, FlexButton).

The following example shows a Flex Button control in the QTP Repository Editor:

Application hierarchy



When you run a test and interact with a Flex control, QTP does the following:

1. Identifies the test object class that your Flex object maps to.
2. Reads the current state of the object and stores its list of description properties and values.
3. Assigns a unique ID to use to identify the object in the scripts. QTP can use one or more of the description properties that might be based on the appearance of the Flex control (such as a Button's label) or on the Flex `id` of the control.
4. Records operations that you perform on the object by using the appropriate test object method.

When a test runs, QTP stores component properties as name-value pairs. QTP only stores properties that are needed for identification in the object repository. These properties include `id` and `index`. You can use checkpoints, data tables, and other methods to store and retrieve other object properties, such as the value of a `TextInput` control's text or the color of a `Button` control's background.

## About the application hierarchy

In the object repository, objects are stored in a hierarchy. The root of the hierarchy is the browser. The next element is the Flex application. Under that are the containers and controls that make up the application.

In the QTP Object Repository and Keyword View, a tree-like structure represents the levels of containers in the application. In general, the hierarchy follows the way the application appears visually. In the MyApp Flex application, for example, for a Button labeled Submit inside a TabNavigator's view labeled Change Settings, the hierarchy appears like this:

```
Browser
|
Application ("MyApp")
|
Panel ("Change Settings")
|
Button ("Submit")
```

In QTP scripts, you use dot-notation syntax to access an object, and a dot separates each level. To refer to the Button, you use a path like the following:

```
Browser("Home").FlexApplication("MyApp").FlexPanel("Change Settings").
    FlexButton("Submit")
```

For information on how a QTP script shows Flex applications, see “About the testing script” on page 13.

Although containers, such as HBox or Panel, are contained in a TabNavigator container, the TabNavigator itself does not appear in the hierarchy. To improve script readability, some Flex containers are omitted from the hierarchy if the container has no impact on the testing.

The application hierarchy is also called the display hierarchy, because it generally represents what you see on the screen in a tree-based layout. The display hierarchy is not always the same as the automation hierarchy, which is the hierarchy of objects as they are referred to in the QTP scripts. Sometimes containers that appear on-screen (for example, with a border or in a window) are part of the display hierarchy, but not part of the automation hierarchy.

In general, if a container does not appear on the screen, it does not appear in the testing hierarchy. For example, a stand-alone HBox container is often used to lay out content, but it has no visual representation and does not appear in the test scripts. The exception to this is when the HBox is a child of TabNavigator. In this case, the HBox is needed to uniquely identify its contents, and therefore it is part of the hierarchy. For more information about using the TabNavigator and other containers, see “Working with containers” on page 21.

# Recording tests

To record a test in QTP, you must compile the application's SWF file and generate the HTML wrapper files. The main wrapper file defines the application as an object on the page and embeds it so that Flash Player is invoked when the page is requested. You cannot request the application's SWF file directly. You can write the HTML wrapper yourself, or generate it with the compiler.

You point QTP to the wrapper file that embeds the application SWF file.

You can use Adobe Flex Builder or the command-line compilers to generate the SWF file for you. After you pre-compile the SWF file, deploy the SWF file and the HTML wrapper files, plus any additional assets such as Runtime Shared Libraries (RSLs), graphics, or other SWF files, in a directory on the web server.

To record a test, you either specify that QTP runs the test on the currently open browser or specify the location of the application as a URL. In the second case, QTP launches the browser and passes it the request URL that you specified. You determine which method to use in the Record and Run Settings dialog box.

In general, you should use a web server to serve the HTML wrapper to the QTP client. You cannot request the HTML wrapper file on the file system.

To have QTP run the test on an application that is currently running in an open browser window, select the Record And Run Test On Any Open Browser option. To have QTP open a browser and request a URL, select the second option, as the following example shows:



When you play back the test, QTP opens the browser and requests the same URL.

Be sure to specify an HTML file or other wrapper file type that embeds the application's SWF file. You should not request the SWF file directly.

## About the testing script

QTP records the test in an object level view called the Keyword View, and a text-only view of steps called the Expert View. You can edit each view, and QTP updates the other one. For example, if you remove a step in the Expert View, QTP removes the step from the Keyword View. The same is true for adding steps or adding and removing checkpoints.

The Expert View represents each step in the test as an action in VBScript. VBScript uses dot-notation to separate parent from child objects. Each line ends with an action, followed by parameters that define that action.

The following example shows a mouse click in the Expert View:

```
Browser("Main Page").FlexApplication("MyApp").FlexButton("b2").Click
```

The follows example shows the Expert View syntax for Flex applications:

```
Browser("document_title").FlexApplication("app_name").test_object("name").  
operation [parameters]
```

The *document\_title* property is defined by the <TITLE> tag in the <HEAD> block of an HTML page. For a Flex application, the default value of the <TITLE> tag in an HTML wrapper is the URL to the MXML file. If page does not have a title, QTP uses `Browser`.

The *app\_name* property is the same as the ID property of the SWF file embedded in the HTML page's <object> tag.

The *test\_object* is the QTP test object that the Flex control corresponds to. For example, a Button control's QTP test object is `FlexButton`. This object takes a *name* property, which is the QTP name for the object as it appears in the object repository. This name is usually based on its appearance (such as a label), and is sometimes the same as the Flex `id` in the Flex MXML source code.

The *operation* property is the event dispatched by the object that QTP records. This property can take zero or more parameters. The number and type of parameters depend on the type of control that is dispatching the event. They can be a key that was held down during the operation, an index of a list entry, or the text entered in a text field. For example, when a user enters characters in a FlexTextArea object, the text that the user entered is shown after the operation. If the user entered the number 42 in a FlexTextArea object, the Expert View shows a statement similar to the following example:

```
Browser("Main Page").FlexApplication("MyApp").FlexTextArea("ta1").Input  
    "42"
```

If the user selects text in a FlexTextArea object, QTP records a Select operation. This operation takes two comma-separated parameters. The first parameter is the starting character position, and the second parameter is the ending character position of the selection. If the user selected the first four characters in a FlexTextArea object, the Expert View shows a statement similar to the following example:

```
Browser("Main Page").FlexApplication("MyApp").FlexTextArea("ta1").Select  
    0,3
```

## Identifying objects

Test objects are the building blocks of the test scripts in QTP. Each object used in the test is stored in the object repository. Flex controls are mapped to these QTP test objects.

When you record a test, QTP adds each object that you interact with to the object repository and references that object by its test object name in the script. The identifier that QTP uses is the name property of the test object in the repository. This property is sometimes the same as the Flex `id` that the Flex developer specifies in the Flex application source code. For example, a ComboBox control with a Flex `id` of "myCombo" is stored in the object repository with the name `myCombo`. In QTP scripts, you reference it as the following example shows:

```
FlexComboBox("myCombo")
```

However, Flex objects are not always identified in the script by their Flex `id` properties. In many cases, the test object name matches what you see on the screen. The QTP test object name of a FlexButton object, for example, is the value of the `label` property of the object and not the `id` property. For example, the following FlexButton is identified in QTP object repository as "Click Me":



In the QTP expert view, you refer to that FlexButton object as the following example shows:

```
FlexButton("Click Me")
```

If the FlexButton object has no label, QTP uses the ToolTip of the Button for the test object name. If there is no label and no ToolTip, QTP uses the source code's `id` property. If the Flex developer did not specify an `id`, QTP uses the index. If you omit all of these, QTP uses its own numbering scheme to identify the object.

The ordering of properties to determine the test object name varies depending on the object. Other objects whose test object names are not the same as the Flex `id` properties, include FlexAccordionHeader, FlexButtonBarButton, FlexCheckBox, FlexLink, FlexPopUpButton, FlexRadioButton, FlexScrollThumb, FlexSimpleButton, FlexSliderThumb, and FlexTab.

## About operations

When you interact with a Flex application while recording, your actions are stored in the QTP script. These actions, such as the click on a button, define the narrative of the test script. They are known in QTP as operations. In Flex, they are known as events. Each operation defines an interaction with a control in the Flex application. You generally use a combination of operations to perform complex interactions with Flex applications, such as filling out a form and submitting it or dragging several objects and dropping them in a particular location.

Flex controls such as button define many events. These events include common ones like `click`, `showToolTip`, and `focusIn`, but also events that are rarely or never invoked directly by the user, such as `creationComplete`, `render`, and `invalid`. Flex events also include very granular actions such as `mouseMove`, `keyDown`, and `focusOut`.

The most commonly-used Flex events are available for use in QTP as operations. However, QTP does not record all events. QTP records the semantically important gestures, or operations that are logically atomic. That is, recording a click event, rather than the combination of the `mouseDown` and `mouseUp` events; or recording a ComboBox's `select` event, rather than a `mouseDown`, `open`, `drag`, `mouseUp`, and `close` event sequence.

When creating a test, not all events that can be recorded are recorded. For example, the `mouseMove` event for most controls is not recorded by default. If it were, the script would include a new step for every mouse movement, which would quickly overwhelm the test. You can instruct QTP to play back this event by adding it manually to the test script. To see a list of available operations for each Flex component, see the QTP Object Type Information document.

In some cases, QTP plays back some of these events as a sequence of smaller events. Click, for example, is played back as a `mouseDown` and `mouseUp` event pair, or `keyDown` and `keyUp`, depending on how it was invoked.

QTP stores a set of parameters for each operation that define aspects of the operation. These parameters include what keys were held down when the mouse was clicked or the X and Y position of the target in a drag and drop operation. Operation parameters also record where the interaction originated—for example, from the keyboard or the mouse.

When recording operations on Flex controls, QTP records the way in which the operation is carried out. QTP records whether events were caused by the mouse or keyboard, as playback can be different for each case. For example, when a button is clicked using the mouse, the following events are dispatched: `mouseDown`, `mouseUp`, and `click`. However, if the button was clicked by having the spacebar pressed when the button had focus, the events dispatched are `keyDown`, `keyUp`, and `click`.

In order to make the scripts more readable, the Flex plug-in minimizes the number of event parameters recorded in a script. To reduce recording common information, QTP has a default parameter for each operation. The default value for the interaction type of a Button click is “mouse”. If an operation parameter is the same as its default value, then QTP does not record it. For example, instead of recording:

```
FlexButton("buttonId").Click "mouse"
```

QTP only records:

```
FlexButton("buttonId").Click
```

The default action is the mouse click because most of the time a `FlexButton` click is done with the mouse and not the keyboard.

# Using checkpoints

Checkpoints let you examine objects and fail or pass the test based on whether the object contains the value you expected. During the test, QTP compares the current value against the expected value in the checkpoint. If the values do not match, then the test fails. If they do match, the test passes.

The Flex QTP plugin supports the following checkpoint types:

- Bitmap
- Standard
- Table

The Flex QTP plugin does not support the following checkpoints:

- Accessibility
- Database
- Image
- Page
- Text
- XML

## Standard checkpoints

Standard checkpoints let you examine the properties of controls. You can examine any Flex object in the object repository. The testable properties include both information about the object's state (such as a `TextArea`'s text property) and information about the object's appearance (such as the `bgcolor` or style property or the `height` and `width` properties).

The available properties are a subset of the Flex control's full property set, but they include most commonly used properties. For a complete list of properties, see the `tea.html` document.

The following example shows part of a script that uses the `Check` method, which verifies whether the actual value of an item matches the expected value:

```
Browser("FlexStore").FlexApplication("flexstore").FlexCanvas("Support").  
  FlexForm("index:4").Check CheckPoint("index:4")
```

For more information, see the [QuickTest Professional Object Model Reference](#).

## Bitmap checkpoints

When using Bitmap Checkpoints, ensure that you account for differences in the appearance of Flex objects. These differences can be due to tab focus on a test object or to the object's reaction to a mouseOver event. If you add a bitmap checkpoint on a region of the application that has an object with focus (and has the halo effect around that object), you must ensure that your object has focus during the test.

The following example shows the differences among a FlexButton test object with no focus, a FlexButton with focus, and a FlexButton with the mouse over it:



FlexImage objects never visibly have focus, so this is not a concern if you use a Bitmap Checkpoint for checking images.

## Table checkpoints

Table checkpoints let you examine properties in List-based controls and containers. You can use table checkpoints with all of the Flex List-based controls, which include the List, DataGrid, TileList, and Tree controls. You can also use table checkpoints with all containers, which generate a list of automationValues for all of the children they contain.

## Using common methods and properties

Flex test objects support the methods and properties that are common to all QTP test objects. The following methods are the common ones:

- CaptureBitmap
- Check
- CheckProperty
- ChildObjects
- GetROProperty
- GetTOProperty

The following properties are the common ones:

- Exist
- Object

You can use the common methods and properties in the Keyword or Expert views. The following example uses the `CheckProperty()` method to check whether the `visible` property of the `myList` control is set to `false` in Expert View:

```
Browser("My Page").FlexApplication("MyApp").FlexCheckBox("myList").  
    CheckProperty "visible", false
```

For information about each of these methods and properties, see the [QuickTest Professional Object Model Reference](#).

## Playing back Flex tests

To play back a test, you do not need to have the browser open. QTP starts the browser for you. You must have a generated HTML wrapper that you request from QTP. You should not request the application's SWF file directly.

In some cases, the playback may not work as expected. This is most often due to synchronization. For example, an effect may take longer to play back than QTP expects, which means that while the effect is playing, QTP may be trying to move to the next step in the script. Another problem might be if you use asynchronous methods of acquiring data, such as a web service call. The script continues even if you have not yet received data from an web service operation.

Solutions to these problems include putting in waits or checking if an object exists before continuing. For more information, see “Resolving playback synchronization issues” on page 24.



This topic describes advanced concepts when using Mercury QuickTest Professional to test Flex applications.

## Contents

Working with containers .....	21
Working with Repeater objects.....	22
Working with data-driven and List-based controls.....	22
Troubleshooting .....	22
Limitations of automated testing with Flex.....	29

## Working with containers

In general, Mercury QuickTest Professional (QTP) reduces the amount of detail about nested controls in the testing script. It removes containers that have no impact on the results of the test or on the identification of the controls from the script. This applies to containers that are used exclusively for layout, such as the HBox, VBox, and Canvas containers, except when they are being used in ViewStack, TabNavigator, or Accordion containers. In these cases, they are added to the hierarchy to provide navigation.

When using nested containers, you should be wary of ID conflicts. It is possible for containers with multiple tabs (such as Accordion and TabNavigator containers) to have the same label on more than one tab. These containers derive their IDs from the tab labels, therefore, there could be overlapping IDs.

QTP does not record layout containers in the scripts unless you execute an event on that container. This keeps the test objects from becoming too deeply nested, which makes the testing scripts less readable.

For more information on the application hierarchy, see “About the application hierarchy” on page 11.

# Working with Repeater objects

Repeater objects are visible in the QTP object hierarchy and scripts, and contain the child objects that they have created. In an actual Flex application, these child objects are children of the main container and not the Repeater object. In QTP table checkpoints, both models are accounted for; the child objects are visible as children of the container and the Repeater object.

# Working with data-driven and List-based controls

To work properly with automation, a custom item renderer object for a List-based control must be data-driven. For generating table checkpoints, data is pushed onto a single item renderer, and then values are queried. If an item renderer is not data-driven, the values generated are incorrect.

If data that is displayed in a List-based control changes across tests, you should use an index-based approach for recording. Value-based recording fails in this situation.

# Troubleshooting

There are some common problems and resolutions when you use QTP to test your Flex applications.

## General troubleshooting

You can solve some common problems by ensuring the following information about your environment and application:

- ActiveX version of the debugger version of Flash Player 9 (version 296) is installed.
- Flex/QTP plug-ins are installed.
- The application was compiled using Flex 2.0 with the new frameworks.swc file that supports automated testing.
- The application is running in Microsoft Internet Explorer.
- The application is loaded through an HTML page, with an `<object>` tag that has an `id` attribute set. The `id` attribute contains no periods or hyphens.
- The application is loaded either from a web server or locally from a trusted SWF file.

If QTP does not record interactions with your Flex application, you should try to determine if the plug-in is installed. The following problems are the most common:

- Flash Player is not the right version.
- Internet Explorer is not the right version.
- Internet Explorer has plug-ins turned off.
- Plug-in was not installed.
- DLLs that the Flex plug-ins are dependent on do not exist in the system. These include MSVCR71.DLL and MSVCP71.DLL.

For information on required versions of applications, see “Quick start” on page 8.

You can determine the problem in the following ways:

- View a list of installed plug-ins.
- View the log files. For more information, see “Logging with Flash Debug Player” on page 23.

## Logging

To help determine the source of problems, you can view Flash Debug Player and QTP log files.

### Logging with Flash Debug Player

Flash Debug Player writes plug-in and ActionScript errors to the flashlog.txt file. The default location of this file is C:\Documents and Settings\*user\_name*. To enable logging, you must configure mm.cfg. For more information, see “Configuring the debugger version of Flash Player” in *Building and Deploying Flex 2 Applications*.

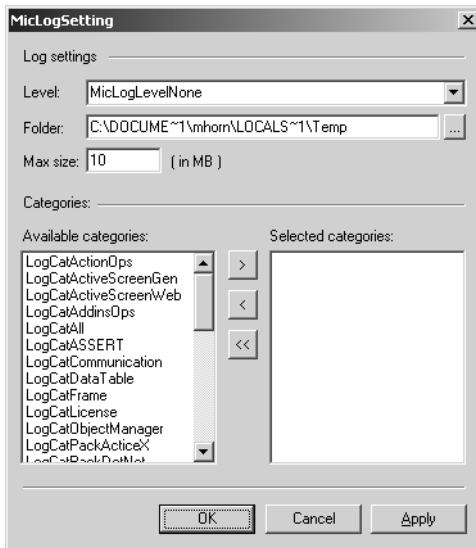
### Logging with QTP

You can configure QTP logging to log QTP errors related to application testing. You then view the QTP log file for possible sources of problems.

#### **To turn on logging of the QTP log file:**

1. Open a command line dialog box.
2. Go to the QTP bin directory. The default location is C:\Program Files\Mercury Interactive\QuickTest Professional\bin.

3. Run MicLogSetting.exe. The MicLogSetting dialog box appears:



4. From the Level drop-down list, select MicLogLevelDebug2 as the level.
5. In the Folder field, select a folder where the log file is saved.
6. From the Available Categories list, select LogCatPackTEA, and click the right arrow button to move it to the Selected Categories list.
7. Click OK to save your changes to the QTP log levels.
8. Run QTP. Record interactions with your Flex application as usual.
9. Close QTP.
10. Examine the QTP log file for potential problems.

NOTE

The QTP log file can be very large. You might want to restrict its use to when you are experiencing errors that you cannot resolve in other ways.

## Resolving playback synchronization issues

In some cases, the playback of a test script is not synchronized with what appears on the screen. For example, QTP does not wait for effects to finish or for data results to be returned before testing an object. This can result in QTP recording errors such as the following:

- Your scripts play back inconsistently.
- You experience an “Unspecified Error”.

- Your scripts are failing in a spot where they should not be.

There are several solutions to these timing issues, which are described in the following sections.

## Adding waits to scripts

You can use the `Wait` statement to pause a script's playback for any number of seconds. You pass `Wait(num_seconds)` as an argument to the operation you want to pause in the script.

The following example waits 10 seconds before clicking the Milk `FlexCheckBox` control:

```
Browser("My Page").FlexApplication("MyApp").FlexCheckBox("Milk").Click  
    Wait(10)
```

For more information on using the `Wait` statements, see the QTP documentation.

## Checking if objects exist

You can also check if the object exists before executing an operation on that control. To do this, you can use the `Exist` statement. The following example checks whether the Milk `FlexCheckBox` control exists before clicking on it:

```
If Browser("My Page").FlexApplication("MyApp").FlexCheckBox("Milk").Exist  
    Then  
    Browser("My Page").FlexApplication("MyApp").FlexCheckBox("Milk").Click  
End If
```

You can also check for the value of the `Exist` common property on the control. For more information on using the `Exist` statement or the `Exist` common property, see the QTP documentation.

## Slowing global playback times

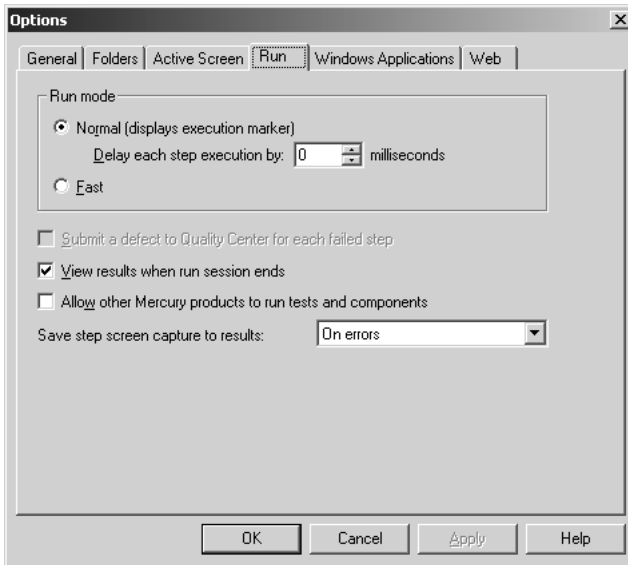
You can configure QTP to play all events slower by setting a global execution delay. This should be considered a last resort. You should first try to insert wait statements before troublesome points in the test, as described in “Adding waits to scripts” on page 25. In very large testing scripts, increasing the time it takes for each event to fire, even by a fraction of a second, can dramatically increase the total run time of the test. Therefore, you should try to find the lowest possible value that lets your scripts run correctly.

If you do choose to add a global delay, you should configure QTP to delay each step in your script by some small number of milliseconds, and then gradually increase this amount until you do not experience any errors.

### To add a delay to each step in your test:

1. Select Tools > Options.

2. In the Options dialog box, select the Run tab.



3. Select the Normal option for the Run mode. This is the default option.
4. Increase the amount of the Delay Each Step Execution By option to some number of milliseconds. The default value is 0.
5. Click OK to save your changes.
6. Rerun your test. If you still experience problems, increase the amount of the delay and rerun the test.

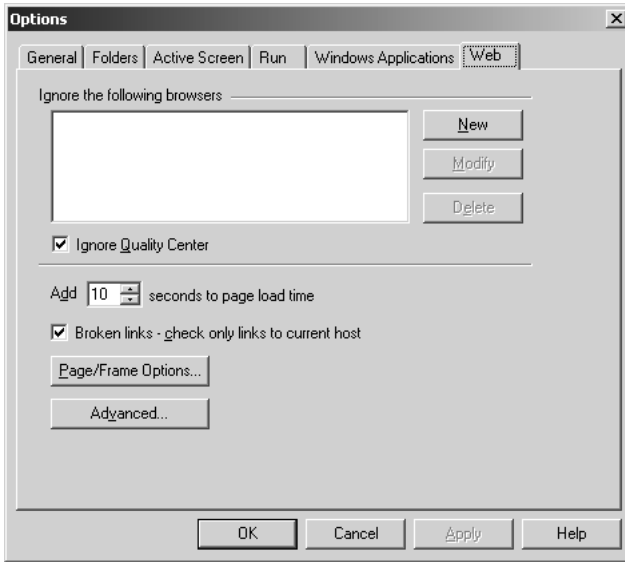
## Delaying startup test time

When you change an application, or recompile the application when QTP launches it in the browser, you should consider delaying the time that the tests start after the browser launches. If you immediately start a test but the application has not finished instantiating all of its objects, QTP might record failures.

### To delay the start of the test:

1. Select Tools > Options.

2. In the Options dialog box, select the Web tab:



3. Increase the amount of the Add *n* Seconds To Page Load Time option to a number large enough to ensure that your application compiles before QTP starts running the test. The default value is 10.

## Making a test fail but continue to the end

You may encounter situations where you cannot make the test fail or pass from a script and still continue to the end. Specifically, this has been reported for tests that change a formula within a Data Table.

To write a QTP method that causes a test to fail but continue to the end, you create a keyword, as the following example shows:

```
Public Function WidthGreaterThan(test_object, minWidth)
actual = test_object.GetROProperty("width")
    a = CInt(actual)
    b = CInt(minWidth)
    If a > b Then
        Msg = "WidthGreaterThan", "Width is greater than: " & minWidth
        Reporter.ReportEvent micPass, Msg
    Else
        Msg = "Width " & actual & " is not greater than " & minWidth
        Reporter.ReportEvent micFail, "WidthGreaterThan", Msg
    End If
End Function

RegisterUserFunc "FlexButton", "WidthGreaterThan", "WidthGreaterThan"
Browser("FlexStore").FlexApplication("flexstore").FlexCanvas("Products").
    FlexButton("Remove from cart_11").WidthGreaterThan 10
Browser("FlexStore").FlexApplication("flexstore").FlexCanvas("Products").
    FlexButton("Remove from cart_11").WidthGreaterThan 1099
Browser("FlexStore").FlexApplication("flexstore").FlexCanvas("Products").
    FlexButton("Remove from cart_11").WidthGreaterThan 100
```

# Limitations of automated testing with Flex

The Flex integration with QTP includes the following limitations:

- Flex test objects do not show up in the Object Identification (Tools > Object Identification) dialog box. Therefore, you generally do not configure the way QTP gets an object `id` or customize which properties QTP stores. You can configure the properties that Flex uses to identify an object in the TEAFlex.xml file.
- Web settings (Tools > Web Event Recording Configuration) do not apply to Flex applications. This is where you can specify the sensitivity of mouse events (for example, instruct QTP to record a mouseDown and then a mouseUp operation, rather than a click operation).
- Flex test objects are not in the QuickTest Object Model Reference (Help > QuickTest Professional Help > Contents tab > QuickTest Object Model Reference). This reference shows all the testable methods and properties, examples, and a list of identification properties. You can refer to the tea.html file that describes all the Flex test objects, their events, and event values.

