FlexMonkey 1.0 User Guide

Version 0.1 Updated x/x/x

FlexMonkey 1.0 User Guide
Eric Owens

Contents

1. Introduction to FlexMonkey

FlexMonkey Capabilities

Built on the Adobe Flex 3 Automation Platform, FlexMonkey is an easy-to-use tool for testing Adobe Flex and Air applications.

Recording

With FlexMonkey, you can run your Flex application, interact with it, and record those interactions. FlexMonkey stores your interactions in a Test folder that you can see and edit in the FlexMonkey Console.

Organizing Tests

FlexMonkey allows you to organize your *Test* folders into *TestCase* folders, and your *TestCase* folders into *TestSuite* folders, in accordance with XUnit test organization.

Assertions

You can add assertions to your FlexMonkey test that check your application's response when the test runs.   Assertions can check the values of a component's properties or styles, or you can capture the bitmap of a visible component and compare it to actual bitmap when your test runs.

Playback

Once you have recorded a test and added assertions, FlexMonkey can play the test, and you can watch as your application is exercised. You can playback individual *Tests*, *TestCases*, or *TestSuites*. FlexMonkey has robust result reporting, and when your test run is completed, FlexMonkey allows you to browse the results

Generating ActionScript TestSuites

FlexMonkey can generate a Fluint/FlexUnit-compatible Actionscript version of your *TestSuite* that you can run from ANT. When you run from ANT, FlexMonkey will create jUnitReport-compatible output that you can integrate with other jUnitReports.

Testing AIR applications

You can test AIR applications with FlexMonkey, but the Flex 3 Adobe Automation platform imposes some limitations. Recording and playback will only work for the main application window and pop-ups from there. Sub-windows and their pop-ups are ignored. Also, File dialogs are ignored.

Adobe is improving the automation system in Flex 4 to cover sub-windows, and FlexMonkey will support Flex 4 in a future release.

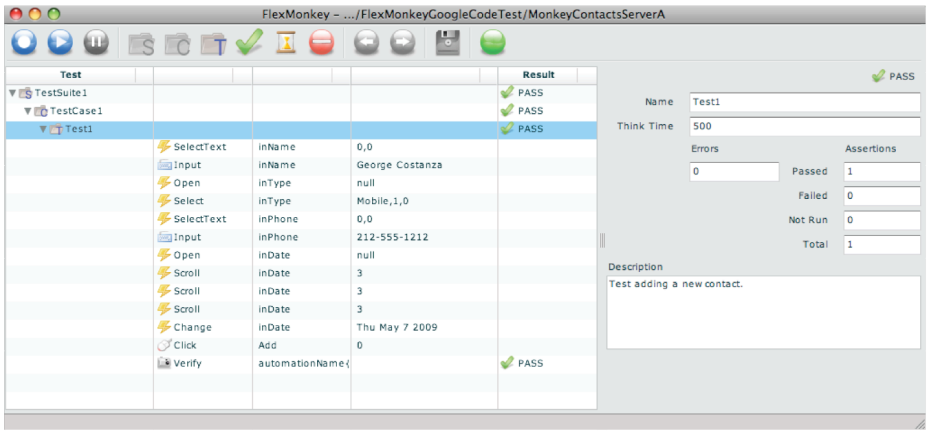
FlexMonkey Main Windows

When you use FlexMonkey with a Flex application, it has 1 main window, the FlexMonkey Console, and your application runs in either the Target SWF Window or the Browser.

If you are testing an AIR application, and you have linked FlexMonkey with the AIR application, you'll have the FlexMonkey Console and your application window.

FlexMonkey Console

All interactive use-modes for FlexMonkey use the FlexMonkey Console (Figure 1-1). If you are running FlexMonkey generated Actionscript tests from ANT, you won't use the console. For more details on the FlexMonkey Console, please see *Chapter 2: FlexMonkey Console*.



FlexMonkey - .../FlexMonkeyGoogleCodeTest/MonkeyContactsServerA

Test				Result
TestSuite1				PASS
TestCase1				PASS
Test1				PASS
	SelectText	inName	0,0	
	Input	inName	George Costanza	
	Open	inType	null	
	Select	inType	Mobile,1,0	
	SelectText	inPhone	0,0	
	Input	inPhone	212-555-1212	
	Open	inDate	null	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Change	inDate	Thu May 7 2009	
	Click	Add	0	
	Verify	automationName{		PASS

PASS

Name: Test1

Think Time: 500

Errors		Assertions
0	Passed	1
	Failed	0
	Not Run	0
	Total	1

Description: Test adding a new contact.

Figure 1-1. The FlexMonkey Console

FlexMonkey Target SWF Window

When you run your application directly under FlexMonkey, as opposed to running your application in a Browser, you'll see FlexMonkey's Target SWF Window (Figure 1-2).

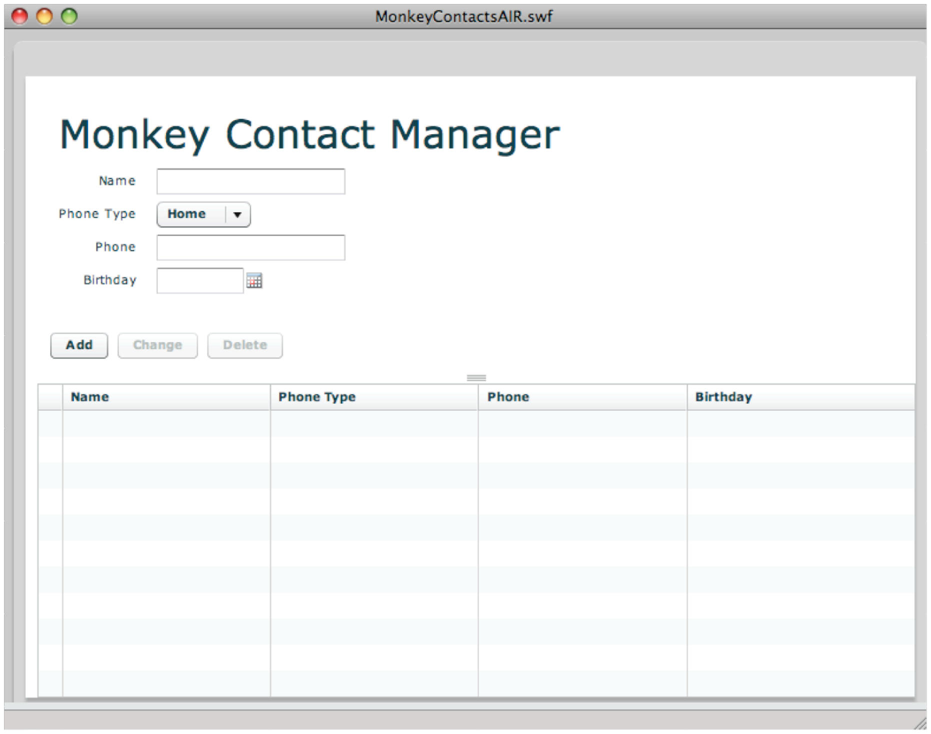
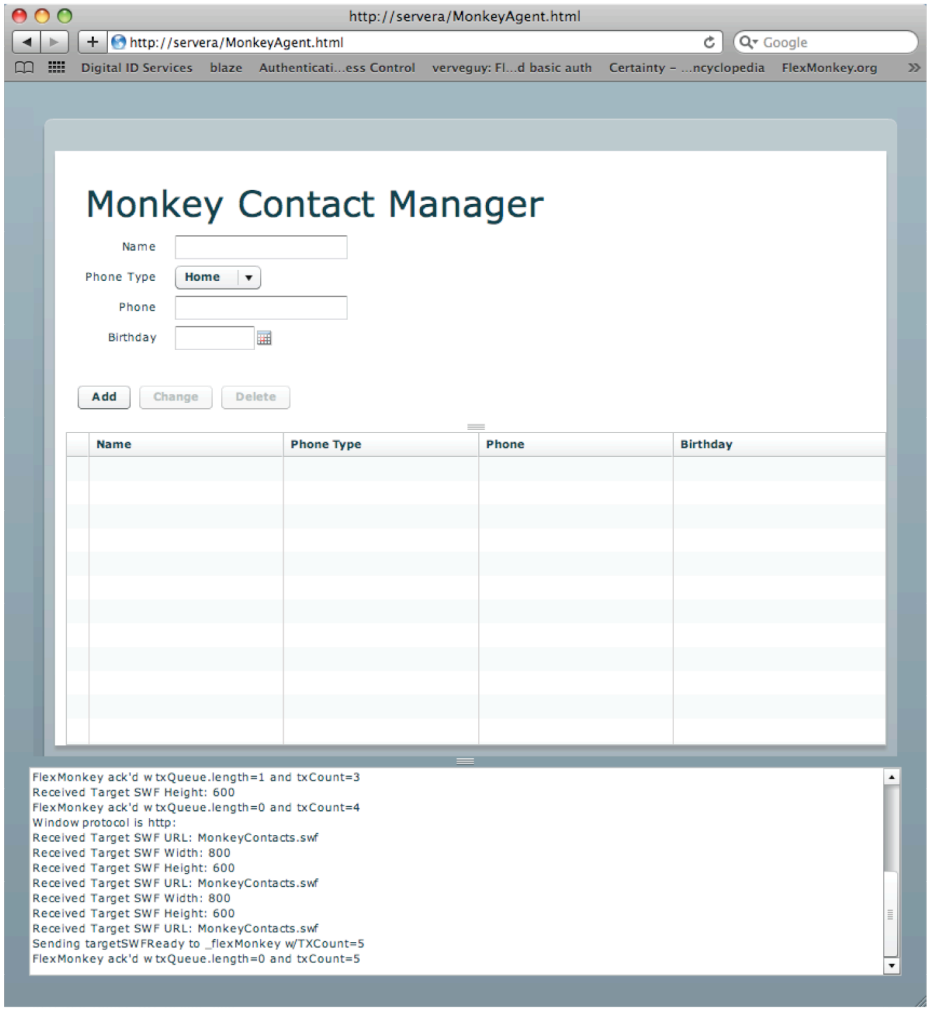


Figure 1-2. MonkeyContacts.swf loaded in the Target SWF Window

MonkeyAgent in the Browser

When you run your application in a Browser, you'll navigate to the MonkeyAgent, and the MonkeyAgent will load your application (Figure 1-3). The MonkeyAgent has an output window where you'll see a log of the agent's communication with the FlexMonkey Console.

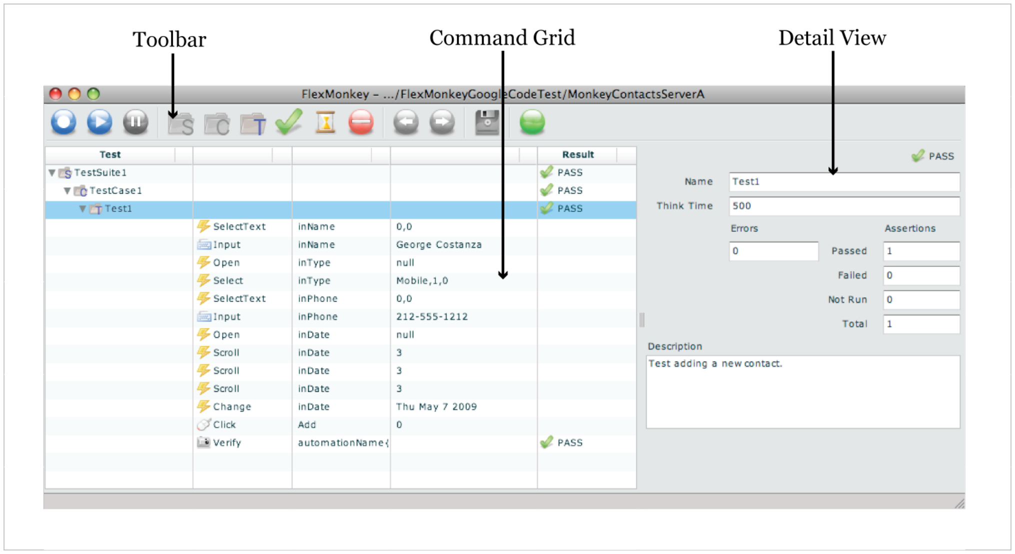


```
FlexMonkey ack'd w txQueue.length=1 and txCount=3
Received Target SWF Height: 600
FlexMonkey ack'd w txQueue.length=0 and txCount=4
Window protocol is http:
Received Target SWF URL: MonkeyContacts.swf
Received Target SWF Width: 800
Received Target SWF Height: 600
Received Target SWF URL: MonkeyContacts.swf
Received Target SWF Width: 800
Received Target SWF Height: 600
Received Target SWF URL: MonkeyContacts.swf
Sending targetSWFReady to _flexMonkey w/TXCount=5
FlexMonkey ack'd w txQueue.length=0 and txCount=5
```

Figure 1-3. MonkeyContacts.swf loaded in the MonkeyAgent

2. FlexMonkey Console

The FlexMonkey Console has three parts, the Toolbar, the Command Grid, and the Detail View (Figure 2-1).


Toolbar
Command Grid
Detail View
FlexMonkey – .../FlexMonkeyGoogleCodeTest/MonkeyContactsServerA

Test				Result
TestSuite1				PASS
TestCase1				PASS
Test1				PASS
	SelectText	inName	0,0	
	Input	inName	George Costanza	
	Open	inType	null	
	Select	inType	Mobile,1,0	
	SelectText	inPhone	0,0	
	Input	inPhone	212-555-1212	
	Open	inDate	null	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Change	inDate	Thu May 7 2009	
	Click	Add	0	
	Verify	automationName{		PASS

PASS

Name: Test1

Think Time: 500

Errors: 0

Assertions: Passed 1, Failed 0, Not Run 0, Total 1

Description: Test adding a new contact.

Figure 2-1. The three parts of the FlexMonkey Console

Toolbar

The FlexMonkey Toolbar appears above the Command Grid in the FlexMonkey Console (Figure 2-2).

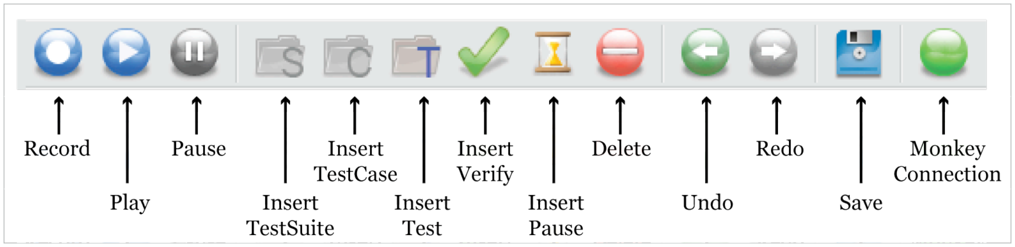


Figure 2-2. The FlexMonkey Toolbar

Record Button

The Record button toggles recording on and off. When recording is on, the center of the button will glow red.

Play Button

The Play button toggles playing on and off. When playing is on, the center of the button will glow green. Playing must begin at a TestSuite, TestCase or Test. While a test is playing, you can stop it by clicking the Play button, but you will need to select a TestSuite, TestCase, or Test before clicking Play again. If you need to pause playing, click the Pause button (see next).

Pause Button

The Pause button toggles pause on and off. When the monkey is playing a test, clicking the Pause button will momentarily halt play. When playing is paused, the center of the Pause button will glow yellow, and the center of the Play button will continue to glow green. To stop pausing, just click the Pause button again.

Insert TestSuite Button

The Insert TestSuite button adds a new, empty TestSuite to your project. To add a TestSuite, select a row in the Command Grid and click the Insert TestSuite button.

Insert TestCase Button

The Insert TestCase button adds a new, empty TestCase to your project. To add a TestCase, select a row in the Command Grid and click the Insert TestCase button.

Insert Test Button

The Insert Test button adds a new, empty Test to your project. To add a Test, select a row in the Command Grid and click the Insert TestCase button.

Insert Verify Button

The Insert Verify button adds a new Verify command to your test. A Verify command can contain assertions about the application you are testing. To add a new Verify command, select a row in a Test and click the Insert Verify command button.

Insert Pause Button

The Insert Pause button adds a new Pause command to your test. A Pause command causes a test to wait for a specifed duration before proceeding with the next command of the test. To add a new Pause command, select a row in a Test and click the Insert Pause button.

Delete Button

The Delete button removes items from the Command Grid. To delete an item, select it in the Command Grid and click the Delete button.

Undo Button

The Undo button undoes the last action taken in the Command Grid. To undo, click the Undo button.

Redo Button

The Redo button redoes the last action taken in the Command Grid. To redo, click the Redo button.

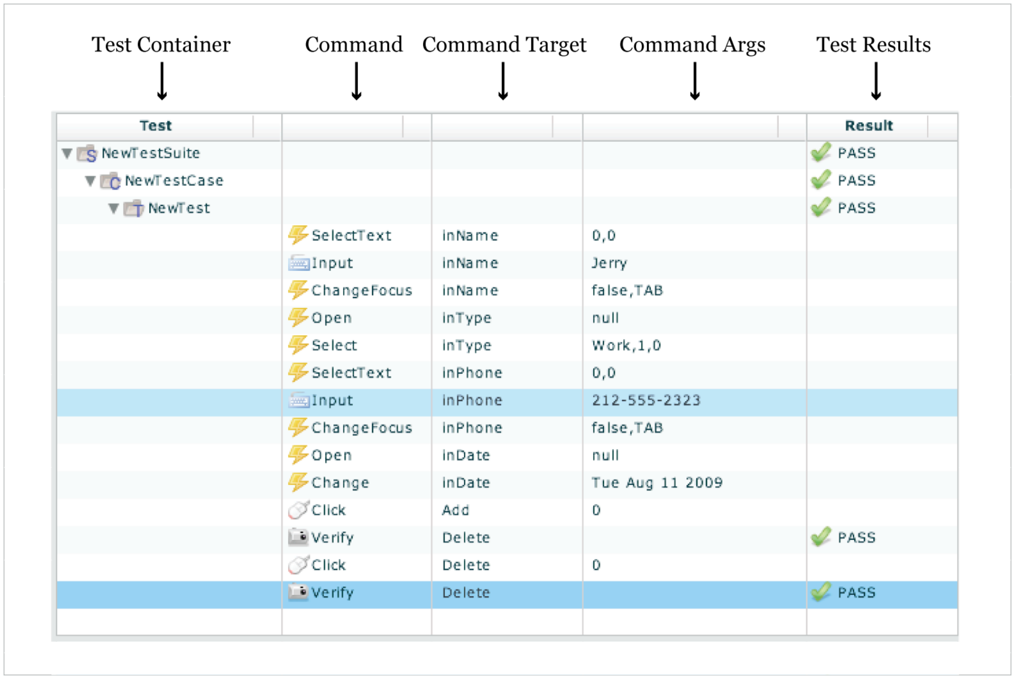
Save Button

The Save button saves the TestSuites in the Command Grid to the monkeyTestSuites.xml file in your project directory. To save, click the Save button.

Monkey Connection Indicator
If you are using the MonkeyAgent or the MonkeyLink, the Browser Connection indicator will glow yellow if FlexMonkey does not detect the presence of the agent or link, and it will glow green if it does detect the agent or the link. If you are using the Target SWF Window, the Browser Connection indicator will be disabled and gray.

Command Grid

The FlexMonkey Command Grid appears just below the Toolbar in the FlexMonkey Console (Figure 2-3).



Test				Result
NewTestSuite				✓ PASS
NewTestCase				✓ PASS
NewTest				✓ PASS
	SelectText	inName	0,0	
	Input	inName	Jerry	
	ChangeFocus	inName	false,TAB	
	Open	inType	null	
	Select	inType	Work,1,0	
	SelectText	inPhone	0,0	
	Input	inPhone	212-555-2323	
	ChangeFocus	inPhone	false,TAB	
	Open	inDate	null	
	Change	inDate	Tue Aug 11 2009	
	Click	Add	0	
	Verify	Delete		✓ PASS
	Click	Delete	0	
	Verify	Delete		✓ PASS

Figure 2-3. The FlexMonkey Command Grid

The Command Grid has 5 columns that show the basic facts about your TestSuites. A row in the grid can be one of the following:

- TestSuite
- TestCase
- Test
- UIEventMonkeyCommand
- PauseMonkeyCommand
- VerifyMonkeyCommand

Test Container Column

If a row is a TestSuite, TestCase, or a Test, the Test Container column displays its name.

Otherwise, the column is left blank.

Command Column

If a row is a UIEventMonkeyCommand, the Command column displays the name of the command.

Otherwise, the column is left blank.

Command Target Column

If a row is a UIEventMonkeyCommand or a VerifyMonkeyCommand, the Command Target column displays the name of the UIComponent that is the target for that row's command.

Otherwise, the column is left blank.

Command Args Column

If a row is a PauseMonkeyCommand, the Command Args column displays the duration of the pause.

If a row is a UIEventMonkeyCommand, the Command Args column contains the arguments that FlexMonkey will send to the Automation Manager to facilitate playback.

Otherwise, the column is left blank.

Test Result Column

All rows in the Command Grid can display results information. Take a look at *Chapter 7: Test Playback* for more information on results reporting.

Detail View

The FlexMonkey Detail View appears just to the right of the Command Grid in the FlexMonkey Console (Figure x). There is a different detail view for each type of Command Grid row entry. For more information on each of the Detail Views, please see *Chapter 5: FlexMonkey Test Elements*.

3. Project Properties

Creating a New Project

When you use FlexMonkey, you create a project for each Application you want to test. Typically, you will locate this project in the same directory as your application's Flex Builder project.

A FlexMonkey project consists of two files and a directory. The file *monkeyTestSuites.xml* contains the tests that you create in the FlexMonkey Console. The file *monkeyTestProject.xml* contains your project preferences. The directory *snapshots* contains the bitmap images of your application's components that are recorded when you add assertions to your tests.

To create a new project, pull down FlexMonkey's File Menu, and select *New Project*. You'll be presented with a Directory browser. Navigate to the directory where you would like to store your project files, and click *Select*.

Opening an Existing Project

If you've previously created a FlexMonkey project and would like to open it, pull down the File menu and select *Open Projec*t. Navigate to the project you would like to open and click *Select*.

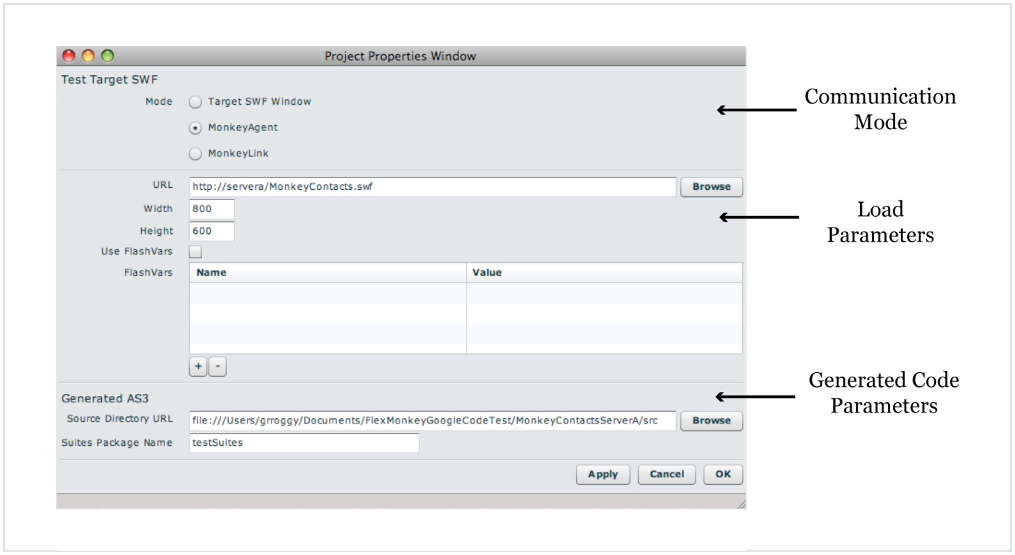
Project Properties Window

After you've selected a directory for your new FlexMonkey project, the Project Properties Window will appear (Figure 3-1). You can also navigate to this window at any time by pulling down FlexMonkey's Project Menu and selecting *Properties*.

There are two main sections in the Project Properties, the Test Target SWF, and the Generated AS3. At the very bottom of the window are three buttons:

- *Apply*
- *Cancel*
- *OK*

As you make choices and changes to your Project Properties, you can click *Apply* or *OK* at anytime to see the effect of the change. Anytime you click *Apply* or *OK*, your changes are saved to the file *monkeyTestProject.xml* in your project directory. When you click *Apply*, the property values are applied to your project, but the Project Properties window remains open. If you click *OK*, the property values are applied to your project, and the window closes. If you click *Cancel*, or just close the window, any changes you entered since that last *Apply* will be ignored.



Test Target SWF

Mode	Target SWF Window [✓ MonkeyAgent] MonkeyLink
URL	http://servera/MonkeyContacts.swf
Width	800
Height	600
Use FlashVars	☐
FlashVars	

Name	Value

Generated AS3

Source Directory URL	file:///Users/grroggy/Documents/FlexMonkeyGoogleCodeTest/MonkeyContactsServerA/src
Suites Package Name	testSuites

Figure 3-1. The FlexMonkey Project Properties Window

Test Target SWF

The Test Target SWF section is allows you to set how the FlexMonkey Console will communicate with the application-under-test and, if you choose to load your application into FlexMonkey, the parameters for the load.

Mode

There are three options when you select Mode:

- Target SWF Window
- MonkeyAgent
- MonkeyLink

For more information on these choices, please see *Chapter 4: FlexMonkey Setup*.

The first two options, Target SWF Window and MonkeyAgent, are the load options. If you select one of these, you'll have some more options:

- Target SWF URL
- Width
- Height
- Use FlashVars
- FlashVar Grid

Target SWF URL

If you are having FlexMonkey load your application for testing, then you need to tell the monkey what application swf to load. You can type in a relative or absolute URL, or, if you will be loading your application swf with a file URL, you can click *Browse* and navigate to the application in the browse dialog.

Only use a relative URL if you are using the MonkeyAgent. In that case, the URL is relative to the directory where you have located the agent.

Width

This property specifies how wide your application's display area should be in pixels.

Height

This property specifies how high your application's display area should be in pixels.

Use FlashVars

If your application's HTML wrapper passes parameters as FlashVars, you will need to specify them in your project properties. Check the *Use FlashVars* check box, and then add the name-value pairs to the FlashVar Grid below the check box.

FlashVar Grid

If you've checked the *Use FlashVars* check box, you can add name-value pairs to the FlashVar Grid. To add a new pair, click the + button and then edit the grid. To remove a previously entered pair, select it in the grid and click the - button.

These name-value pairs are added to Application.application.parameters and are accessible by your application when it loads.

Generated AS3

FlexMonkey can generate an ActionScript version of the TestSuites in the FlexMonkey Console. The Source Directory and Suites Package Name control aspects of the generation.

Source Directory URL

The Source Directory URL is the directory in your filesystem where FlexMonkey should put the top-level package folder for your TestSuites. By convention, this is typically placed just under the *src* folder in your Flex Builder project directory.

Suites Package Name

This is the name of the top-level package folder that will be placed at the Source Directory URL.

4. FlexMonkey Setup

There are a number of ways you can run your application when you use FlexMonkey. You can

- load your application into FlexMonkey's Target SWF Window,
- load your application into FlexMonkey's MonkeyAgent, or
- link FlexMonkey's MonkeyLink directly into your application.

Which one you use depends on the requirements of your application and your testing needs.

Load vs. Link

The most fundamental choice you make when deciding how to run your application is whether you will load your application into FlexMonkey or link your application with FlexMonkey. Table 4-1 below summarizes the requirements and conditions of loading vs. linking.

	Need Flex Builder Pro License	Must recompile application-under-test	Root Application	HTML wrapper
Load	No	No	FlexMonkey or MonkeyAgent	MonkeyAgent
Link	Yes	Yes	Application-under-test	Application-under-test

Table 4-1. Load vs. Link

Loading is the easiest option. You don't need a Flex Builder Pro license and you don't need to recompile your application. However, when you run your application after loading, you must be aware that your application will not be the root application.   Rather, FlexMonkey will be the root.

When FlexMonkey is the root, some code in your application may work differently than when the application is launched normally, including:

- Calls to Application.application
- Loading of relative-path URLs
- AIR's File.applicationDirectory and .applicationStorageDirectory
- HTML wrapper-related code
- ApplicationDomain-dependent code

In some cases, there are workarounds for these issues.

Loading: Application.application workarounds

When your application uses the ActionScript construct Application.application, it will refer to FlexMonkey as the application, not your application.

If you have access to the application source code, you can modify it to address this problem. If you substitute *this.parentApplication* for *Application.application* in your application's source code, you will redirect the access to your application's application.

One exception is if you use FlashVars to initialize your application. Because FlexMonkey's approach to FlashVars is to add them to FlexMonkey's Application.application.parameters object, so you should still access your FlashVars as

Application.application.parameters[myVar].

For more information on using FlashVars with FlexMonkey, see the section on FlashVars.

Loading: Relative URL workarounds

If your application loads files with relative paths, it will load them relative to FlexMonkey's root directory.

If you are using the MonkeyAgent, you can address this problem by co-locating the MonkeyAgent in the same directory as the application.swf.

If you can't co-locate the MonkeyAgent, you can try to address this problem by using absolute URLs in your application source, but if the file you are loading is a SWF, you may run into problems with Application Domain or Security Domain conflicts.   For more information, look here.

Loading: File.applicationDirectory workarounds

If your application is an AIR application, when your application uses the ActionScript construct File.applicationDirectory and File.applicationStorageDirectory, they will point to FlexMonkey's directories.

There are no workarounds currently -- you will need to link FlexMonkey with your AIR application if you use this functionality.

Loading: HTML wrapper workarounds

If you are using the MonkeyAgent to load your application, your application's HTML wrapper will be MonkeyAgent.html. Any custom code in your application's HTML wrapper, including FlashVars or custom javascript you access through ActionScript's ExternalInterface ,will not be available.

You can customize the MonkeyAgent's wrapper to address this problem. If you add any application-specific code from your application's HTML wrapper to the MonkeyAgent's wrapper, your application will have access to the resources it needs.

Loading: ApplicationDomain-dependent code

When you are loading, FlexMonkey or the MonkeyAgent are the parent Application domains. Any class definitions that are included in the parent will be used by your application. So, for instance, since FlexMonkey and the MonkeyAgent are compiled with the 3.3 SDK, your application's classes will be using the 3.3 class definitions.

To workaround this problem, you can recompile your application with the 3.3 SDK, or you can recompile FlexMonkey or the MonkeyAgent from source.

Load the application into Target SWF Window

The easiest way to use FlexMonkey is to load your application directly into the FlexMonkey Target SWF Window. However, if your application needs to be launched from a Web Server, this mode is not for you.

To load your application directly into the FlexMonkey Target SWF Window, you navigate to the Project Properties Window, and set the Communications Mode to Target SWF Window (the default for new projects). Then, FlexMonkey will display a window where the application swf pointed to by the Target SWF URL in the Load Parameters.

Load the application into the MonkeyAgent

You can run your application in a Browser by using the MonkeyAgent. The MonkeyAgent is a Flex application that you launch in the browser. The MonkeyAgent can be local or remote. Typically, you colocate the MonkeyAgent with the application you want to test. After the MonkeyAgent is running, it will communicate with the FlexMonkey Console and load your application as a sub-application of the MonkeyAgent.

In the paragraphs that follow, ‘MonkeyAgent’ refers to a collection of related files. The MonkeyAgent is comprised of:

- MonkeyAgent.html
- MonkeyAgent.swf
- AC_OETags.js

When it says below to place the MonkeyAgent in a directory, it means to place all three files there -- although if AC_OETags.js is already in the directory where you want to put the MonkeyAgent, you won’t need the AC_OETags.js that comes with the MonkeyAgent.

To use the MonkeyAgent, you first need to decide where you will deploy the agent relative to where your application is deployed. Table 4-2 summarizes your choices.

Option	Application Location	MonkeyAgent Location	Must Set Flash Global Security Settings	Must deploy crossdomain policy file
1	Flex Builder Project Directory	Flex Builder Project Directory	No	No
2	file:///	file:///	Yes	No
3	http://myServerA	http://myServerA	No	No
4	http://myServerA	http://myServerB	No	Yes

Table 4-2. MonkeyAgent Deployment Options.

Option 1 is simple. You place the MonkeyAgent in your Flex Builder project directory, alongside your application in bin-debug. Then you use a relative URL for the Target SWF in your project properties, that is, just specify *applicationName.swf*.

Option 2 is very much like option 1, except that the MonkeyAgent is placed on the file system but outside of a Flex Builder project directory. You will need to edit your Flash Global Security settings so that FlashPlayer always trusts the MonkeyAgent's directory. Please see here for more details.

Option 3 has your application and the MonkeyAgent co-located in a web-server directory. Similar to Option 1, you'll use a relative URL for your Target SWF.

Option 4 supports locating the MonkeyAgent on a different server than your application, but if you do so, you'll need to place a crossdomain policy file in your application's directory that grants permission to the MonkeyAgent's domain. Here is a sample crossdomain.xml file:

```
<?xml version="1.0"?>
```

```
<!-- "www.myServerA.com" crossdomain policy file -->
<cross-domain-policy>
   <site-control permitted-cross-domain-policies="master-only" />
   <allow-access-from domain="www.myServerB.com" to-ports="*"/>
</cross-domain-policy>
```

For more information on crossdomain policy files, please see the section on Website controls here.

Run the application in the debugger

You can run your application in the Flex Builder Debugger while it is under the control of FlexMonkey. This setup is a variation of the MonkeyAgent setup. You'll need to have a Flex Builder project for the application you want to test, and you'll need to create a Flex Builder project for the MonkeyAgent source code. If you launch the MonkeyAgent in the debugger, it will communicate with the FlexMonkey Console and load your application. If you set breakpoints in your application and play a FlexMonkey test, the Flex Builder debugger will halt your application as expected. While your application is halted at a breakpoint, FlexMonkey will pause the playback of the test.

To run your application in the Flex Builder Debugger while using FlexMonkey, you first set up a Flex Builder Project for the MonkeyAgent. The easiest way to do this is to download the Easy2BuildMonkeyAgent Flex Builder project from the FlexMonkey googlecode site at

http://flexmonkey.googlecode.com/svn/trunk

Note that you can also run your application in the debugger if you link FlexMonkey with your application as described in the next section.

Link FlexMonkey with your application

You can build the MonkeyLink into your application, and when you run your application, it will communicate with the FlexMonkey Console to record and playback tests.

To link your application with the MonkeyLink,you'll need to have Flex Builder Pro so that you can build a project with the Automation libraries. You'll also need to download the Easy2BuildMonkeyLink Flex Builder project from the FlexMonkey googlecode site.

Then, in your application's Flex Builder project, make the following changes:

In *Project Properties/Flex Build Path/Library Path* add the Easy2BuildMonkeyLink library project you downloaded earlier.
In *Project Properties/Flex Compiler/Additional compiler arguments,* add each of these include statements:

```
-includes com.gorillalogic.monkeylink.MonkeyLink
-include-libraries "../../Easy2BuildMonkeyLink/libs/FlexAutomationLibrary.swc"
-include-libraries "${flexlib}/libs/automation.swc"
-include-libraries "${flexlib}/libs/automation_agent.swc"
-include-libraries "${flexlib}/libs/automation_dmv.swc"
```

Run Generated AS3 tests from ANT

You can use FlexMonkey to create TestSuites, and then have FlexMonkey generate ActionScipt versions of your tests. You can then run these ActionScript tests in a FlexMonkey Target SWF window, or in a Browser. For more information, please see the FlexMonkey Code Generation Guide.

5. FlexMonkey Test Elements

Currently, FlexMonkey has the following test elements:

- TestSuite
- TestCase
- Test
- UIEventMonkeyCommand
- PauseMonkeyCommand
- VerifyMonkeyCommand

The first three, TestSuite, TestCase, and Test, are the hierarchical containers you can see in the FlexMonkey Console. The last three, UIEventMonkeyCommand, PauseMonkeyCommand, and VerifyMonkeyCommand, are used to build tests.

TestSuite

The TestSuite is the highest-level hierarchical container. Typically, you only have one TestSuite per project, but you can have more. A TestSuite contains 1 or more TestCases.

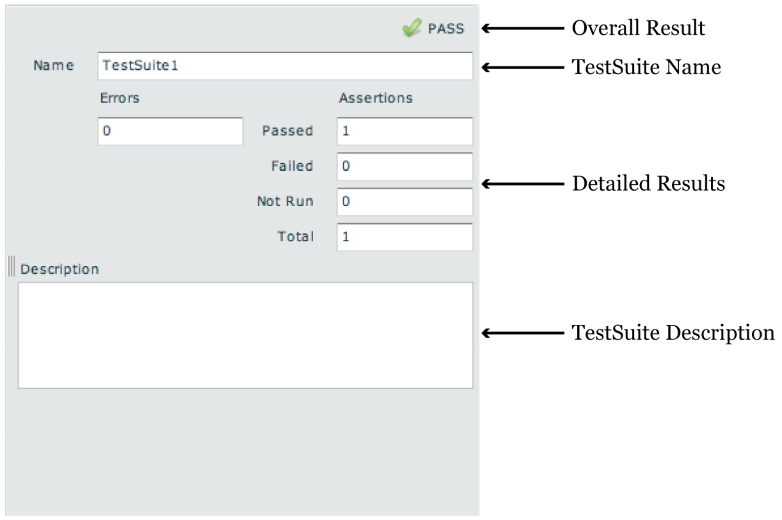
The first field in the TestSuite is the overall result of running the TestSuite. This result is calculated based on all of the TestSuite's sub-results.

The TestSuite also includes fields which summarize the sub-results, including the number of Errors and the disposition of all the Assertions.

For more information on FlexMonkey's results calculations, please see *Chapter 7: Test Playback*.

TestSuite Detail View

When you select a TestSuite in the FlexMonkey Console’s Command Grid, the TestSuite Detail View displays the details of the TestSuite (Figure 5-1).



PASS

Name: TestSuite1

Errors: 0

Assertions

Passed	1
Failed	0
Not Run	0
Total	1

Description

Figure 5-1. The TestSuite Detail View

The TestSuite Detail View allows you to edit the Name field and the Description Field.

TestCase

The TestCase is the middle-level hierarchical container. You use TestCases to organize your Tests into logical groups.

TestCase Detail View

When you select a TestCase in the FlexMonkey Console's Command Grid, the TestCase Detail View displays the details of the TestCase. The TestCase Detail View is identical to the TestSuite Detail View.

Test

The Test is the lowest-level hierarchical container. A Test contains UIEventMonkeyCommands that drive your application's GUI, PauseMonkeyCommands that inject pauses of specific durations into the flow of UIEventMonkeyCommands, and VerifyMonkeyCommands that inject assertions about the state of your application into the flow of UIEventMonkeyCommands.

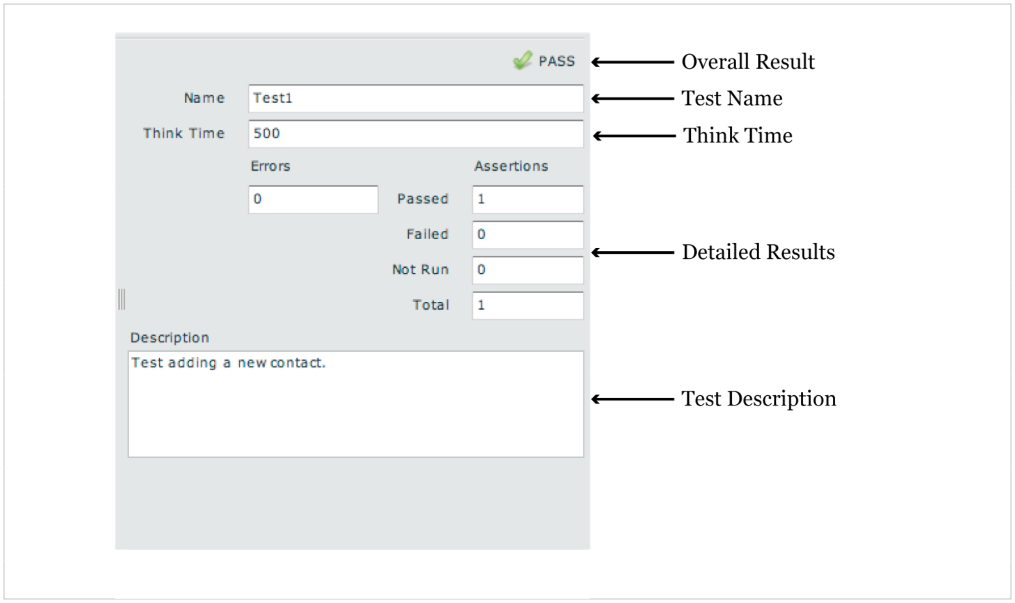
The fields in the Test are nearly identical to the TestSuite and TestCase, but the Test also includes the ThinkTime field.

ThinkTime determines how long FlexMonkey waits between each command when running tests. For more information on ThinkTime, please see *Chapter 6: Creating Tests.*

Test Detail View

The Test Detail View is nearly identical to the TestSuite and TestCase Detail Views, but it also displays the Test's Think Time (Figure 5-2).

The Test Detail View allows you to edit the ThinkTime, as well as the Name and Description fields.



Name	Test1
Think Time	500

Errors		Assertions
0	Passed	1
	Failed	0
	Not Run	0
	Total	1

Description

Test adding a new contact.

Figure 5-2. The Test Detail View

UIEventMonkeyCommand

The UIEventMonkeyCommand captures the data FlexMonkey needs to playback interactions with your application's GUI. Currently, the only way to insert a UIEventMonkeyCommand into a Test is by recording GUI interactions.

The first field in a UIEventMonkeyCommand is its Error Result. The Error Result has a value only for commands that fail to execute. Commands fail to execute when FlexMonkey encounters a problem in trying to play them back. If a command displays an error, the detailed reason for the error will be in the Error Message field.

The rest of the fields in the UIEventMonkeyCommand are divided into two main parts, the Command Target Details and the Command Arguments.

The Command Target Details include

- Target Value
- Target Property
- Container Value
- Container Property.

The Target Value and Target Property pair are filled in by the FlexMonkey Test Recorder when you record an interaction with your application's GUI. They identify the component with which you interacted. The Target Property is the name of the component's property that is being used to identify the component. The Target Value is the value of the Target Property.

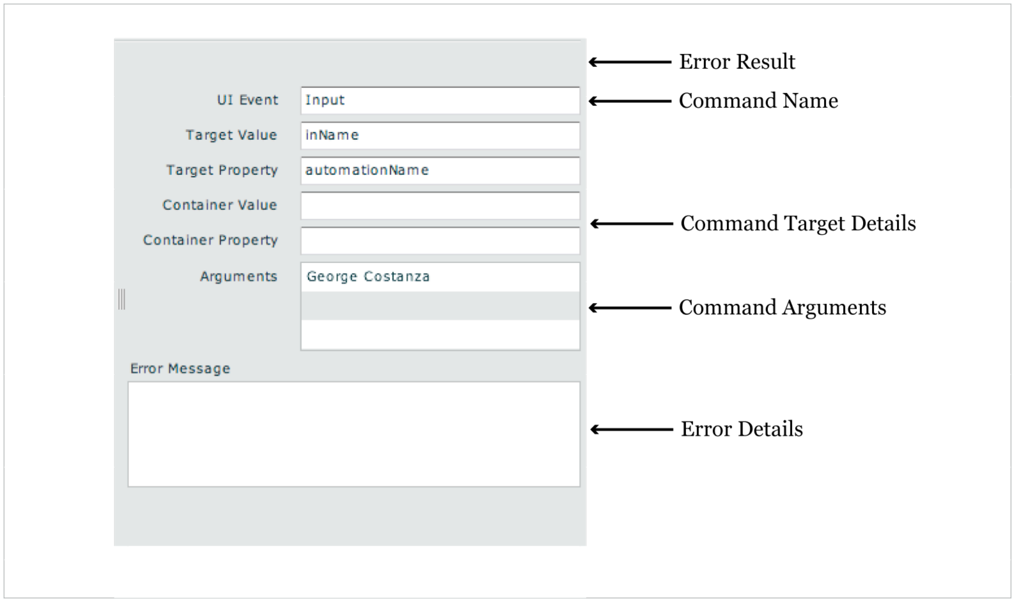
Sometimes, the Target Property and Target Value pair are not sufficient to uniquely identify a Target Component. When this happens, FlexMonkey may play back a UIEventMonkeyCommand on a component different from the one you intended.

You can use the Container Property and Container Value to help disambiguate the Target Component. If you find you need to use these fields, you can choose any component that is in the parent-chain above the intended component in the display hierarchy, and fill in a property/value pair for that containing component.

Currently, you must manually set the Container Property and Container Value if FlexMonkey has a problem finding the intended component for a UIEventMonkeyCommand.

UIEventMonkeyCommand Detail View

When you select a UIEventMonkeyCommand in the FlexMonkey Console's Command Grid, the UIEventMonkeyCommand Detail View displays the details of the command (Figure 5-3).



UI Event	Input
Target Value	inName
Target Property	automationName
Container Value	
Container Property	
Arguments	George Costanza

Error Message

Figure 5.3. The UIEventMonkeyCommand Detail View

UIEventMonkeyCommand ActionScript Syntax

The UIEventMonkeyCommand in the generated ActionScript TestCase has the following constructor syntax:

```
new UIEventMonkeyCommand(commandName:String,
                         targetValue:String,
                         targetProperty:String,
                         containerValue:String = null,
                         containerProperty:String = null,
                         commandArguments:Array)
```

PauseMonkeyCommand

Sometimes, your test needs to wait after a UIEventMonkeyCommand for the application to respond before proceeding. The PauseMonkeyCommand inserts a pause into the flow of UIEventMonkeyCommands and VerifyMonkeyCommands.   The Duration field specifies the pause in mS.

PauseMonkeyCommand Detail View

The only detail available in the PauseMonkeyCommand Detail View is the Pause's duration in mS (Figure 5-4).

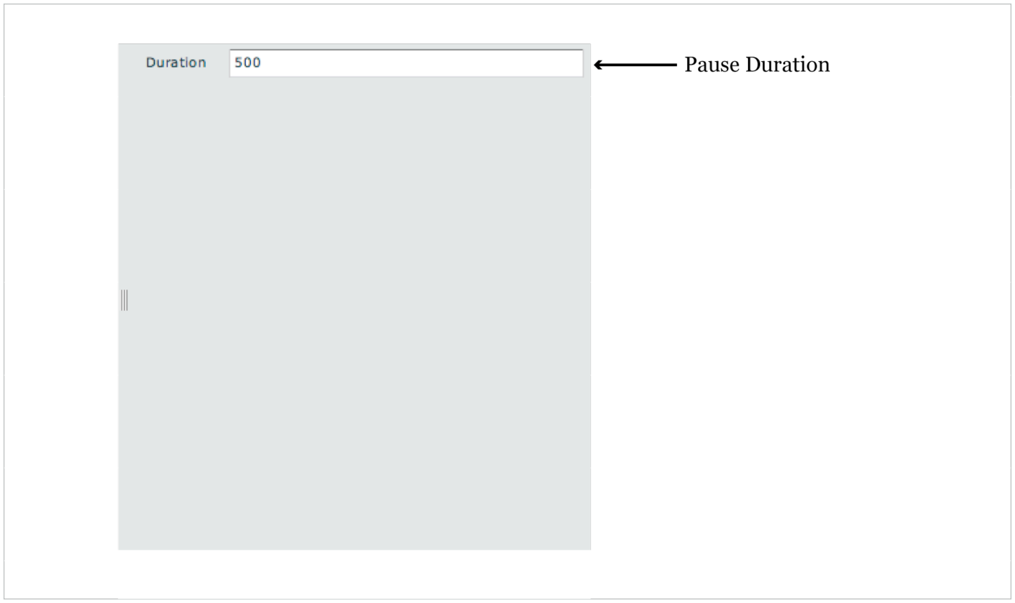


Figure 5-4. The PauseMonkeyCommand Detail View

PauseMonkeyCommand ActionScript Syntax

The PauseMonkeyCommand in the generated ActionScript TestCase has the following constructor syntax:

```
new PauseMonkeyCommand(duration:uint)
```

VerifyMonkeyCommand

The VerifyMonkeyCommand allows you to insert assertions into your tests and have FlexMonkey check these assertions as it plays back your tests.

FlexMonkey has two main types of assertions:

- Bitmap
- Property/Style

The first type of assertion, Bitmap, compares an expected bitmap image of one of the UI components in your application to the actual bitmap image that FlexMonkey encounters when it plays back your test. A VerifyMonkeyCommand can contain zero or one Bitmap assertions.

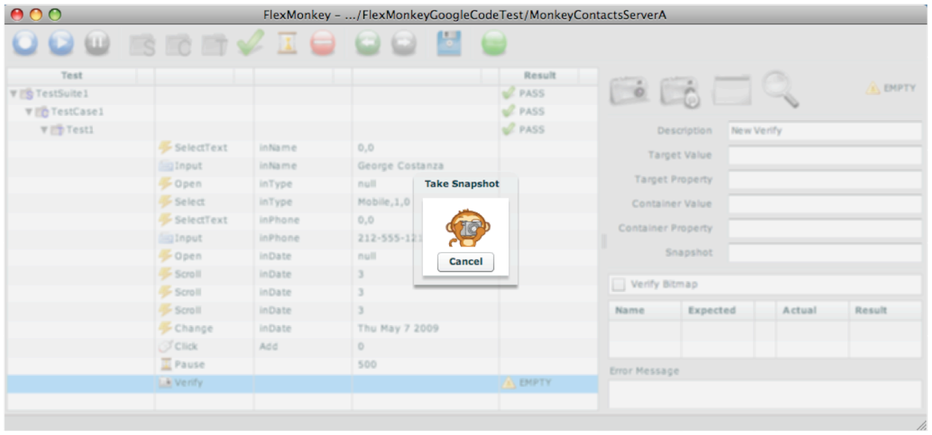
The second type of assertion, Property/Style, compares an expected value for a property or style attribute of one of the UI components in your application to the actual value of that property or style that FlexMonkey encounters when it plays back your test. A VerifyMonkeyCommand can contain 0 or more Property/Style assertions.

VerifyMonkeyCommand Wizard

To add a VerifyMonkeyCommand to a Test, click on the *Insert Verify Button*. When you do, you invoke the VerifyMonkeyCommand wizard. The wizard will walk you through these steps:

- Take Component Snapshot
- Select Component Attributes
- Edit VerifyMonkeyCommand

The first step is to take a component snapshot. When you take a snapshot, you are selecting a component to be the target of the VerifyMonkeyCommand's assertions. When the wizard is in the *Take Snapshot* mode, the FlexMonkeyConsole is unavailable until you either select a component or cancel the snapshot (Figure 5-5).


FlexMonkey – .../FlexMonkeyGoogleCodeTest/MonkeyContactsServerA

Test				Result
TestSuite1				PASS
TestCase1				PASS
Test1				PASS
	SelectText	inName	0,0	
	Input	inName	George Costanza	
	Open	inType	null	
	Select	inType	Mobile,1,0	
	SelectText	inPhone	0,0	
	Input	inPhone	212-555-121	
	Open	inDate	null	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Scroll	inDate	3	
	Change	inDate	Thu May 7 2009	
	Click	Add	0	
	Pause		500	
	Verify			EMPTY

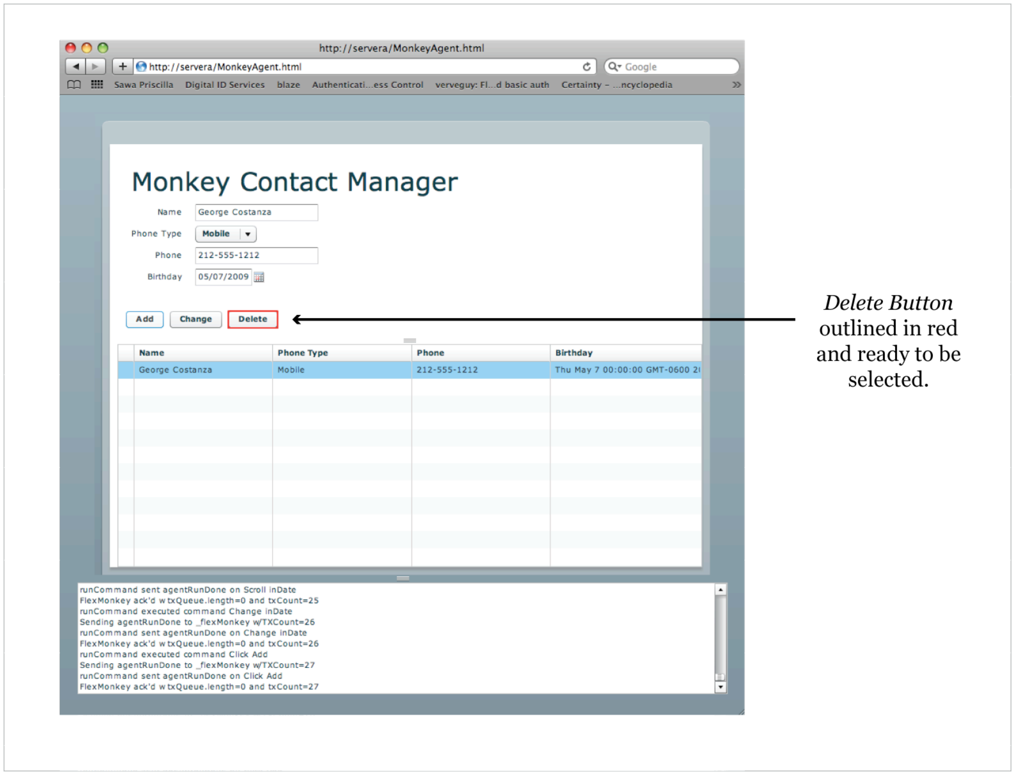
Description: New Verify
Target Value:
Target Property:
Container Value:
Container Property:
Snapshot:
Verify Bitmap

Name	Expected		Actual	Result

Error Message

Figure 5-5. FlexMonkey Console in *Take Snapshot* Mode

While in *Take Snapshot* mode, you can roll your mouse cursor over your application, and FlexMonkey will outline in red the component your cursor is above. If you click the mouse, the outlined component becomes the target of the VerifyMonkeyCommand (Figure 5-6).



Monkey Contact Manager

Name	George Costanza
Phone Type	Mobile
Phone	212-555-1212
Birthday	05/07/2009

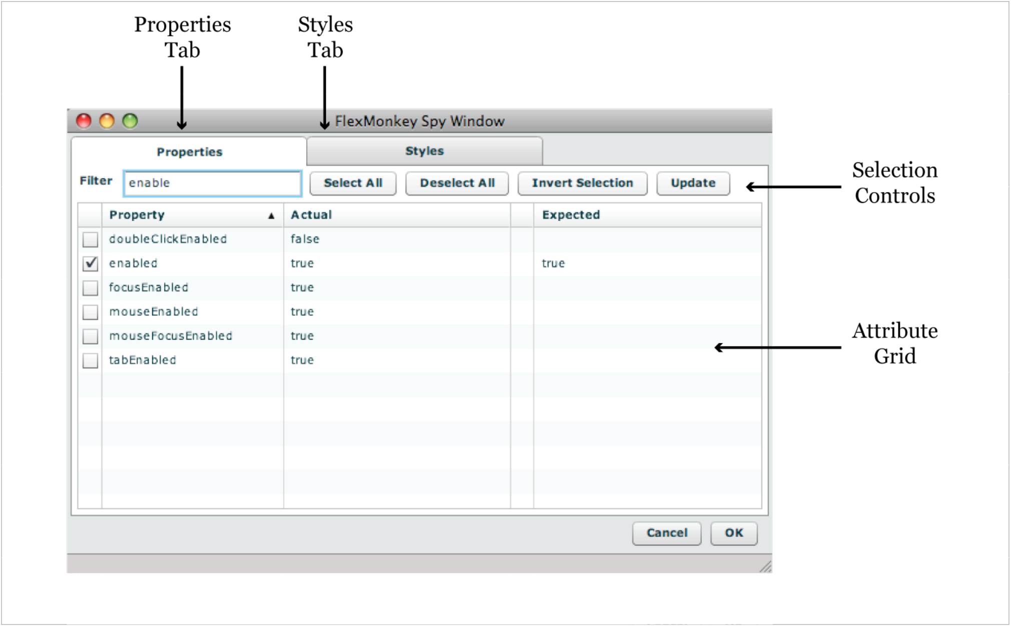
Name	Phone Type	Phone	Birthday
George Costanza	Mobile	212-555-1212	Thu May 7 00:00:00 GMT-0600 2

```
runCommand sent agentRunDone on Scroll inDate
FlexMonkey ack'd w txQueue.length=0 and txCount=25
runCommand executed command Change inDate
Sending agentRunDone to _flexMonkey w/TXCount=26
runCommand sent agentRunDone on Change inDate
FlexMonkey ack'd w txQueue.length=0 and txCount=26
runCommand executed command Click Add
Sending agentRunDone to _flexMonkey w/TXCount=27
runCommand sent agentRunDone on Click Add
FlexMonkey ack'd w txQueue.length=0 and txCount=27
```

Figure 5-6. MonkeyAgent with application in *Take Snapshot* mode

In addition to making the highlighted component the target component of the VerifyMonkeyCommand, FlexMonkey also stores a copy of the component's current visual state as a bitmap in your FlexMonkey project's *snapshot* directory.

Once you have taken a component snapshot, the VerifyMonkeyCommand pops up the FlexMonkey Spy Window (Figure 5-7) and enables you to select Property and Style attributes for assertions.



	Property ▲	Actual		Expected
☐	doubleClickEnabled	false		
✓	enabled	true		true
☐	focusEnabled	true		
☐	mouseEnabled	true		
☐	mouseFocusEnabled	true		
☐	tabEnabled	true		

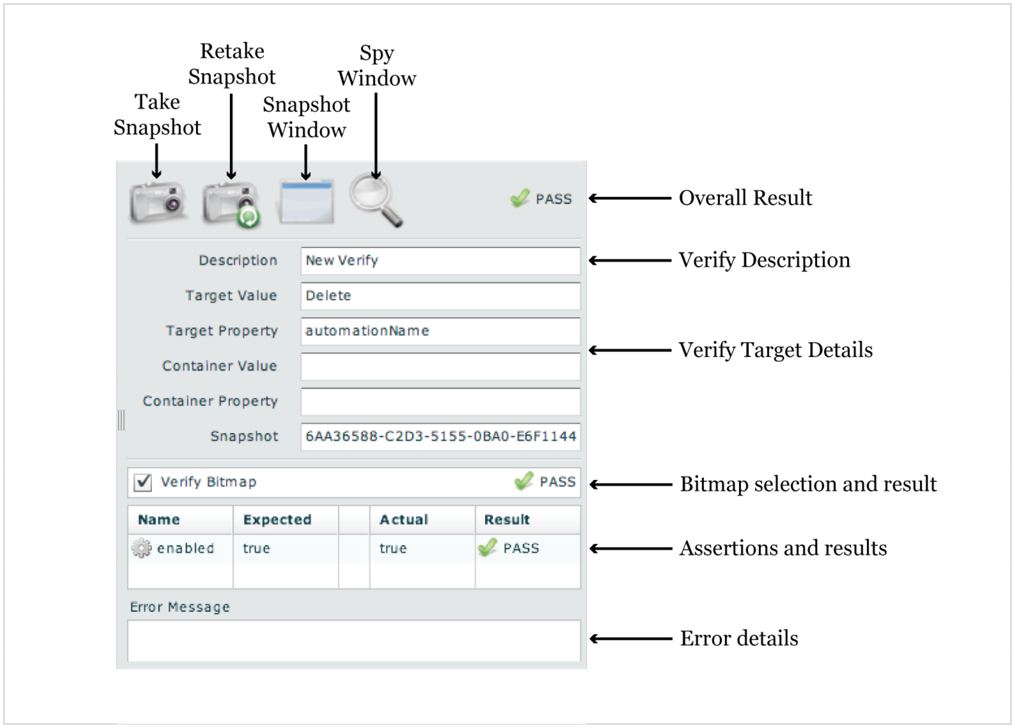
Figure 5-7. The FlexMonkey Spy Window

The FlexMonkey Spy Window has two tabs, one that displays property attributes and one that displays style attributes. Each of these tabs has a set of selection controls that makes it easy to find and select just the attributes you want to make assertions about. The attribute grid in each tab lets you select attributes by checking the box in the far left column. If you do select an attribute, FlexMonkey transfers its current value as displayed in the Actual column to the Expected column.

Once you have selected all the property and style attributes you need in your VerifyMonkeyCommand, click the *OK* button and FlexMonkey transfers your selected attributes to the VerifyMonkeyCommand Detail View. The VerifyMonkeyCommand Detail View is the final stop of the VerifyMonkeyCommand Wizard.

VerifyMonkeyCommand Detail View

The VerifyMonkeyCommand Detail View displays the details and tools required for working with assertions (Figure 5-8).



✓ PASS

Description	New Verify
Target Value	Delete
Target Property	automationName
Container Value	
Container Property	
Snapshot	6AA36588-C2D3-5155-0BA0-E6F1144

☑ Verify Bitmap ✓ PASS

Name	Expected		Actual	Result
enabled	true		true	✓ PASS

Error Message

Figure 5-8. The VerifyMonkeyCommand Detail View

Along the top of this view is a toolbar that contains 4 tools:

- *Take Snapshot* button
- *Retake Snapshot* button
- *Snapshot Window* button
- *Spy Window* button

The *Take Snapshot* button sends FlexMonkey back to *Take Snapshot* mode and lets you choose a different component as the target of the VerifyMonkeyCommand's.

The *Retake Snapshot* button captures a new copy of the VerifyMonkeyCommand's current visual state as a bitmap. This is useful if you are using the VerifyMonkeyCommand's capability to make bitmap comparisons, and need to update the expected bitmap.

The *Snapshot Window* button displays the expected bitmap representation of your VerifyMonkeyCommand's target component, and, if available, the actual bitmap that FlexMonkey encountered when executing the VerifyMonkeyCommand during Test playback. By default, when you click the *Snapshot Window* button FlexMonkey displays the window in *Expected* mode, and the window shows the expected bitmap. If you click in the Snapshot Window anywhere but on the expected component's bitmap, FlexMonkey toggles the display to *Actual* mode and shows the actual bitmap, if available (Figure 5-9).

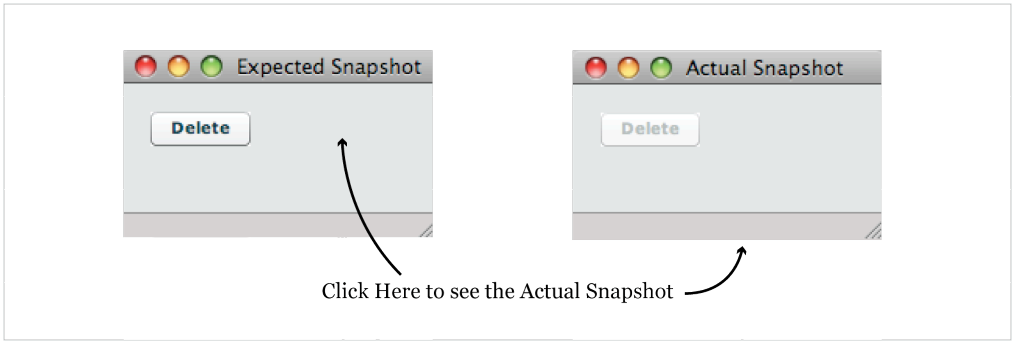
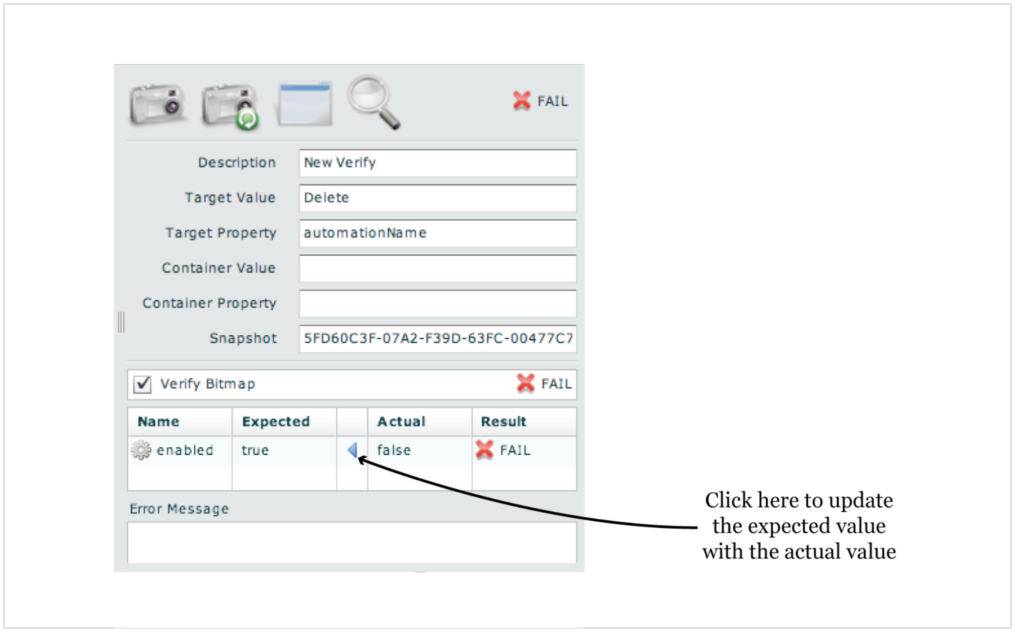


Figure 5-9. Snapshot Window in *Expected* and *Actual* modes

The *Spy Window* button displays the Spy Window and enables you to edit your property and style attribute choices.

Sometimes when you play back a test and your assertions fail, the problem lies with the assertion's expected value and not in the application. In those cases, you may want to update the assertion's expected value with the actual value FlexMonkey encountered at test playback time. After FlexMonkey has run a test, you can browse through your assertions in either the VerifyMonkeyCommand Detail View or the Spy Window. In either view, if FlexMonkey has a failed assertion where expected and actual values are not equal, FlexMonkey makes it easy to update an expected value to an actual value. FlexMonkey will display a small blue arrow in the grid and if you click the arrow, FlexMonkey will update the expected value with the actual value (Figure 5-10).


FAIL

Description	New Verify
Target Value	Delete
Target Property	automationName
Container Value	
Container Property	
Snapshot	5FD60C3F-07A2-F39D-63FC-00477C7

Verify Bitmap ✓ FAIL

Name	Expected		Actual	Result
enabled	true	◀	false	FAIL

Error Message

Click here to update the expected value with the actual value

Figure 5-10. Updating the expected value with the actual value

VerifyMonkeyCommand ActionScript Syntax

The VerifyMonkeyCommand in the generated ActionScript TestCase has the following constructor syntax:

```
new VerifyMonkeyCommand(description:String,
                        snapshot:String,
                        targetValue:String,
                        targetProperty:String,
                        containerValue:String = null,
                        containerProperty:String = null,
                        verifyBitmap:Boolean,
                        attributes:ArrayCollection)
```

The *attributes* ArrayCollection is comprised of AttributeVO elements that have the following constructor syntax:

```
new AttributeVO(attributeName:String,
                namespaceURI:String = null,
                attributeType:String,   // either “property” or “style”
                expectedValue:String)
```

CallMonkeyCommand

The CallMonkeyCommand is only available for use in generated ActionScript TestCases, and must be added to a generated TestCase manually. It allows you to call an arbitrary function at any point during a test.

CallMonkeyCommand ActionScript Syntax

The CallMonkeyCommand in the generated ActionScript TestCase has the following constructor syntax:

```
new CallMonkeyCommand(func:Function)
```

A typical use of the CallMonkeyCommand uses a closure:

```
new CallMonkeyCommand(function():void{Application.application.writeConsole("In
Test And Things Are Happening");})
```

6. Creating Tests

You create Tests with FlexMonkey by recording your interactions with your application, and then adding assertions to those tests with the FlexMonkey Console. You adjust how your tests will be played back by FlexMonkey by adjusting ThinkTime and adding Pauses in the test flow.

Recording

To record a test, first select a Test folder or a command row inside a Test. FlexMonkey will only enable the *Record* button if you are targeting a Test. If you select a Test folder, FlexMonkey will add recorded interactions at the beginning of that Test, even if the Test has some commands in it already. If you select a command row inside a Test, FlexMonkey will add recorded interactions after the command on that row.

To begin the actual recording, click the *Record* button on the FlexMonkey Toolbar and interact with your application. When recording is active, the center of the *Record* button will glow red.

As you interact with your application, FlexMonkey adds commands for each of the interactions to the Command Grid. You can stop recording at anytime by clicking the *Record* button again. When you do, the center of the *Record* button will be white again.

Controlling ThinkTime

When FlexMonkey plays back your Test, it will wait for a ThinkTime duration between each command in the test. You can set the

ThinkTime for the Test by selecting the Test folder and editing the ThinkTime field in the Test Detail View. ThinkTime duration is measured in milliseconds, and it defaults to 500 milliseconds.

Adding Pauses

Sometimes a command in your Test will take longer than the Test's ThinkTime to complete. For instance, after submitting Login credentials, your application makes a call to the server to authenticate the user before displaying the first screen of the application. When this happens, you'll need to insert a PauseMonkeyCommand into your Test so that when the first command after the pause executes, the application will be ready.

To insert a Pause, select the row in the Command Grid just before the row where you would like to insert the Pause, then click the *Insert Pause* button on the FlexMonkey Toolbar.

You can edit the pause duration in the PauseMonkeyCommand's Detail View. The pause duration is measured in milliseconds, and the default pause is 500 milliseconds.

Adding Assertions

You can add assertions to your Test by inserting VerifyMonkeyCommands into the Test flow. Assertions allow you to check that your Application is behaving correctly. For more information on using the VerifyMonkeyCommand, please see *Chapter 5: FlexMonkey Test Elements*.

7. Test Playback

You can playback:

- a single Test
- all the Tests in a TestCase
- all the TestCases in a TestSuite.

To playback a single Test, select the Test in the Command Grid, and click the *Play* button.

To playback all the Tests in a TestCase, select the TestCase in the Command Grid and click the *Play* button.

To playback all the TestCases in a TestSuite, select the TestSuite in the Command Grid, and click the *Play* button.

While the Test is playing, the arrow in the *Play* button will glow green. When the Test is complete, the arrow in the *Play* button will return to white.

You can pause playback at any time by clicking the *Pause* button. While playback is paused, the || symbol in the *Pause* button will glow orange. To resume playback, click the *Pause* button again. The || symbol in the Pause button will return to white.

Currently, FlexMonkey can only begin playback on a Test, TestCase or TestSuite.

Understanding Test Results

FlexMonkey has a number of symbols that it uses to indicate results status:

✱ NOT_RUN
✓ PASS
✗ FAIL
❗ ERROR
⚠ EMPTY

✱ **NOT_RUN**

For an assertion in a VerifyMonkeyCommand, this symbol indicates that FlexMonkey has not played back this command and checked the assertion since the last time its result was reset. FlexMonkey resets results when it first opens a project, and when a user-initiated change is made that effects the validity of a result.

For an overall result in a VerifyMonkeyCommand, Test, TestCase, or TestSuite, this symbol indicates that FlexMonkey has not checked at least one of the assertions below.

✓ **PASS**

For an assertion in a VerifyMonkeyCommand, this symbol indicates that FlexMonkey has checked the assertion and that it was true.

For an overall result in a VerifyMonkeyCommand, Test, TestCase, or TestSuite, this symbol indicates that FlexMonkey has found all the assertions below to be true.

❌ **FAIL**

For an assertion in a VerifyMonkeyCommand, this symbol indicates that FlexMonkey has checked the assertion and that it was false.

For an overall result in a VerifyMonkeyCommand, Test, TestCase, or TestSuite, this symbol indicates that FlexMonkey has found at least one of the assertions below to be false.

❗ **ERROR**

For an overall result in a UIEventMonkeyCommand or VerifyMonkeyCommand, this symbol indicates that FlexMonkey encountered a problem playing back the command. Details on the problem can be found in the command's Error Message field in its Detail View.

For an overall result in a Test, TestCase, or TestSuite, this symbol indicates that FlexMonkey has found at least one of the UIEventMonkeyCommands or VerifyMonkeyCommands below to be an Error.

⚠ **EMPTY**

For a VerifyMonkeyCommand that has no Property/Style assertions and does not have its *Verify Bitmap* check box checked, this symbol indicates that the VerifyMonkeyCommand has no assertions to check.

For an overall result in a Test, this symbol indicates that there are no commands to playback in the Test.

For an overall result in a TestCase or TestSuite, this symbol indicates that there are no non-empty Tests below.