

Adobe Flex Recording with SOASTA CloudTest™

Adobe Flex Recording with SOASTA CloudTest

©2009, SOASTA, Inc. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective companies.

This document is for informational purposes only. SOASTA makes no warranties, express or implied, as to the information contained within this document.

Adobe Flex Recording with SOASTA CloudTest

SOASTA CloudTest™ can automatically record and test Adobe Flex applications and the way that users interact with them.

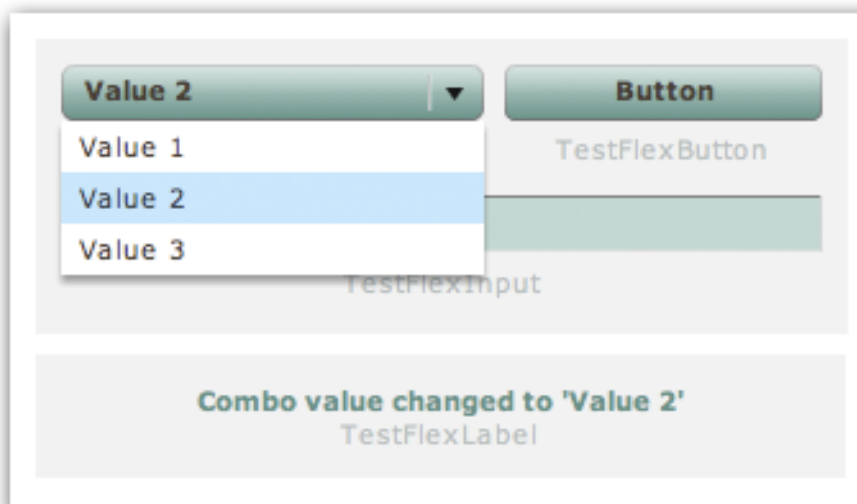
Rather than the simple functional debugging offered within the Adobe Flex SDK or the Flex Builder IDE, imagine Adobe Flex applications as part of much larger tests that involve 1,000s (or many more) of users at once. Using SOASTA CloudTest you can discover how Adobe Flex applications will perform in real-world web sites involving many users from diverse locations.

SOASTA CloudTest's Browser Recording capability can automate the creation of Adobe Flex test cases by recording and presenting Flex actions just as it does other Browser Actions from a target Web application.

Recording browser actions within Adobe Flex applications is achieved by adding simple Action Scripts to existing Flex code. Once an Adobe Flex application is automated for recording using the techniques below, the basics of a Web UI test including Adobe Flex can be created very quickly.

Getting Started with Flex Recording

As an example of Adobe Flex extensibility, SOASTA has created a simple Flex component with four controls (a ComboBox, a Button, a TextInput, and a Label). Changing the ComboBox value, clicking on the Button, or setting the text of the TextInput will change the text of the Label control. Each control ID is displayed in a gray label below it: TestFlexCombo, TestFlexButton, TestFlexInput, and TestFlexLabel.



In order to automate these controls for Flex recording and testing, we will create an Action Script and add a means to identify each control.

The Flex code archive

The procedures below can be applied to any Flex component. However, the following sections discuss how to extend our sample Flex component. Optionally, you can download an [archive of this test application](#) including the compiled object and its code. The FlexAUT.zip archive contains three files:

- FlexAUT.mxml – This file contains the code for the flex sample before adding automation and recording capabilities
- testAPI.as – This file contains the Action Script(s) that automate the component for use with SOASTA CloudTest and its Browser Recorder add-on for the Firefox browser
- FlexAUT.swf – The compiled Flex component prior to SOASTA CloudTest automation

Note: Refer to Adobe Flex documentation for more information about the [ExternalInterface](#) class. You will need the Flex SDK or Flex Builder IDE in order to re-compile the sample component.

Adding Flex Automation Capability

To enable automation through SOASTA, we must add Action Script functions for each of the events that we want to automate for SOASTA CloudTest.

1. Create an action script file, testAPI.as and add the following helper functions for locating the Flex components by id.

```
/**
 * Find a UIComponent using its id attribute, wherever it is in the application
 * @param id id attribute of the UIComponent to return
 * @return the UIComponent corresponding to the id, or null if not found
 */
private function getElementById(id:String):UIComponent {
    return getElementByIdRecursive(id, this);
}

private function getElementByIdRecursive(id:String, root:UIComponent):UIComponent {
    for(var i:int = 0; i<root.numChildren; i++) {
        try {
            var child:UIComponent = UIComponent(root.getChildAt(i));
            if(child.id == id) return child;
            var node:UIComponent = getElementByIdRecursive(id, child);
            if(node!=null) return node;
        }
        catch(e:Error) {
        }
    }
    return null;
}
```

2. Next, add a function to select a value in the combo box by its label.

```
import flash.events.Event;
import mx.events.ListEvent;

/**
 * Select a value in a combo box
 * @param id id attribute of the combo box to use
 * @param label label to select in the combo box
 * @return true if there was no exception, false otherwise
 */
private function selectByLabel(id:String, label:String):Boolean {
    var combo:ComboBox, retval:Boolean = false;
    try {
        combo = ComboBox(getElementById(id));
        combo.selectedItem = label;
        combo.dispatchEvent(new ListEvent(ListEvent.CHANGE));
        retval = true;
    }
    catch(e:Error) {
    }
    return retval;
}
```

3. Then, add a function to click on the button.

```
/**
 * Click on a button
 * @param id id attribute of the button to click
 * @return true if there was no exception, false otherwise
 */
public function clickButton(id:String):Boolean {
    var btn:Button, retval:Boolean = false;
    try {
        btn = Button(getElementById(id));
        btn.dispatchEvent(new MouseEvent(MouseEvent.CLICK));
        retval = true;
    }
    catch(e:Error) {
    }
    return retval;
}
```

4. Add a function to change the value of the text input.

```
/**
 * Set a text input value
 * @param id id attribute of the text input to set the text of
 * @param text the text to set
 * @return true if there was no exception, false otherwise
 */
private function setInputText(id:String, text:String):Boolean {
    var input:TextInput, retval:Boolean = false;
    try {
        input = TextInput(getElementById(id));
        input.text = text;
        input.dispatchEvent(new Event(flash.events.Event.CHANGE));
        retval = true;
    }
    catch(e:Error) {
    }
    return retval;
}
```

```
}
```

5. Expose all of the automation functions that were just added to JavaScript with the following code in `testAPI.as`.

```
**  
* Register all the functions available from JavaScript  
* @return true if the functions were properly registered, false otherwise  
*/  
private function registerTestCallbacks():Boolean {  
    var retval:Boolean = false;  
    if(ExternalInterface.available==true) {  
        ExternalInterface.addCallback("selectByLabel", selectByLabel);  
        ExternalInterface.addCallback("clickButton", clickButton);  
        ExternalInterface.addCallback("setInputText", setInputText);  
        retval = true;  
    }  
    return retval;  
}
```

6. Then we include **testAPI.as** in our MXML file and add the call to **registerTestCallbacks** in the **onInit** handler.

```

<mx:Script>
  <![CDATA[
    import mx.collections.ArrayCollection;

    /* testAPI contains the functions used for automation*/
    include "testAPI.as";

    [Bindable]
    private var comboContent:ArrayCollection;

    private function onInit():void {
      comboContent = new ArrayCollection(['Value 1', 'Value 2', 'Value 3']);
      this.registerTestCallbacks();
    }

    /* Functions triggered on UI component events*/
    private function onTestFlexComboChange():void {
      this.TestFlexLabel.text = "Combo value changed to '" + this.TestFlexCombo.selectedLabel
+ "'";
    }
    private function onTestFlexButtonClick():void {
      this.TestFlexLabel.text = "Button clicked";
    }
    private function onTestFlexInputChange():void {
      this.TestFlexLabel.text = "Text input value set to '" + this.TestFlexInput.text + "'";
    }
  ]]>
</mx:Script>

```

7. At this point, the Flex component is ready to be tested by SOASTA. Compile the above .mxml and .as files to create **FlexAUT.swf**. Add the Flex object to an HTML page with an id of **FlexAUT**.

```

<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  id="FlexAUT" width="336" height="194"
  codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
  <param name="movie" value="bin-debug/FlexAUT.swf" />

  <param name="quality" value="high" />
  <param name="bgcolor" value="#869ca7" />
  <param name="allowScriptAccess" value="sameDomain" />
  <embed src="bin-debug/FlexAUT.swf" quality="high" bgcolor="#869ca7"
    width="336" height="194" name="FlexAUT" align="middle"
    play="true"
    loop="false"
    quality="high"
    allowScriptAccess="sameDomain"
    type="application/x-shockwave-flash"
    pluginspage="http://www.adobe.com/go/getflashplayer">
  </embed>
</object>

```

Creating a Flex Target and Test Clip

Now that we have an Adobe Flex component with automation features, log in to SOASTA CloudTest and navigate to the SOASTA Repository > Target.

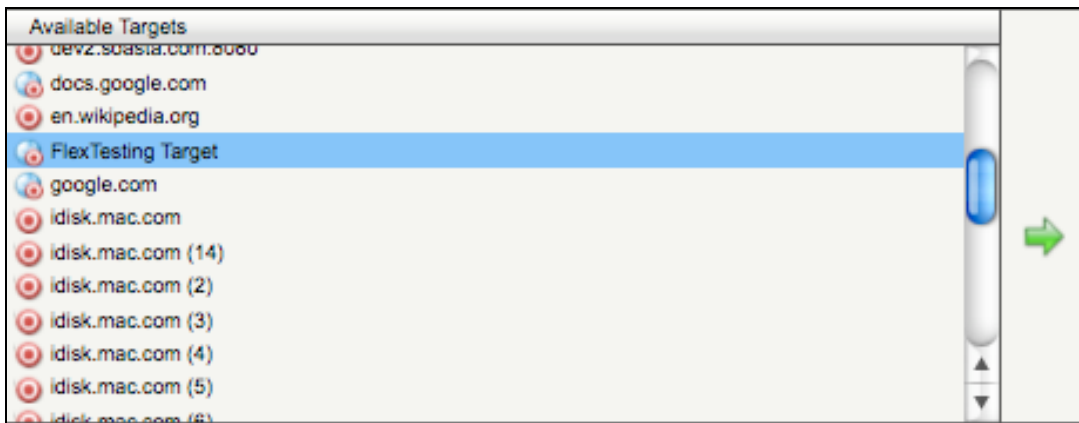
Note: The following steps require a SOASTA CloudTest user ID and access to a CloudTest server, as well as Firefox browser and the SOASTA Browser Recorder add-on.

1. Use SOASTA CloudTest to create a WebUI target that points to the HTML page where you posted the <object> code above.

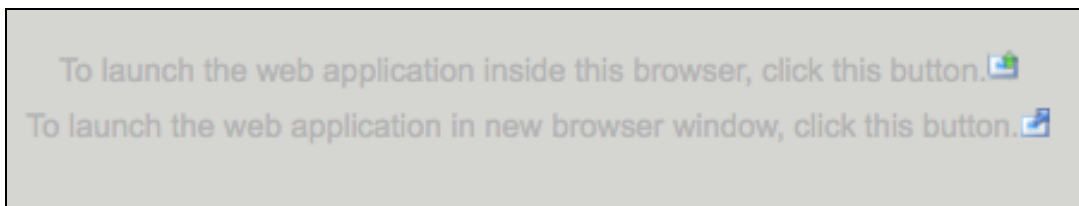
With Repository > Targets selected, click New. The Target Wizard appears. Click Next to get started with target creation. Specify a WebUI target and supply the URL of the HTML page that includes the newly compiled Flex component. Save the target.

2. Use SOASTA CloudTest to create a Test Clip that uses the target created in the prior step.

With Repository > Test Clips selected, click New. The Clip Editor appears with the Available Targets list displayed below. Click the Flex Target and then click the green arrow to populate the lists to the right.

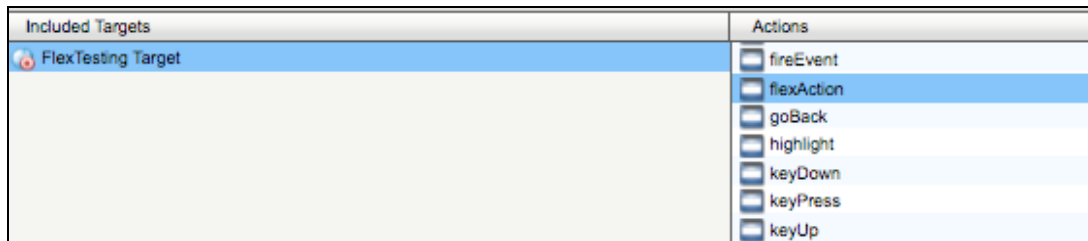


When you do so, the Flex target is added to the Included Targets and the Clip Editor workspace displays the following links.



Since we have a bit more work to do before browser recording works for our flex actions, let's manually add each flexAction to the Clip from the list of generic browser actions.

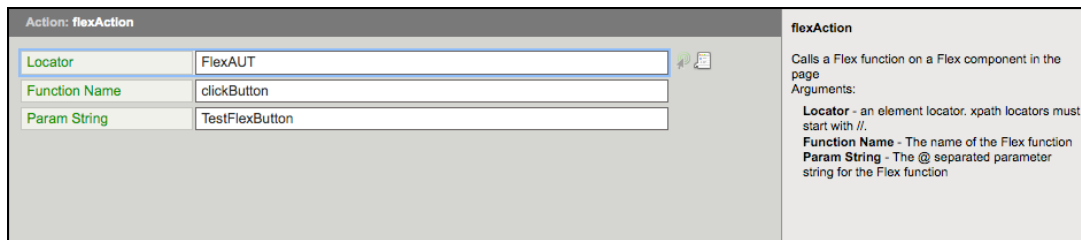
3. To manually add the generic flexAction, select your target in the Included Targets list and then browse to flexAction in the Actions list on the right.
4. Add one flexAction for each of the functions to automate.
 - setInputText
 - selectByLabel
 - getLabelText
 - clickButton



5. Once you are done adding the Flex actions, select the first Flex browser action in the Test Clip and double-click it to open its properties below.



6. For each of the generic flexAction items open in the Browser Action Properties pane below, enter the Locator *FlexAUT*.
7. Then, choose a function name to define. For example, *clickButton*.



8. Enter the Param String value. For example, *TestFlexButton*.

Note: For the selectByLabel and setTextInput functions, which both have two parameters, those parameters are separated by an '@' symbol.

9. Repeat steps 6-8 for each of the flexAction test clip elements.

Action: flexAction		flexAction
Locator	FlexAUT	Calls a Flex function on a Flex component in the page Arguments: Locator - an element locator. xpath locators must start with //. Function Name - The name of the Flex function Param String - The @ separated parameter string for the Flex function
Function Name	selectByLabel	
Param String	TestFlexCombo@Value 2	

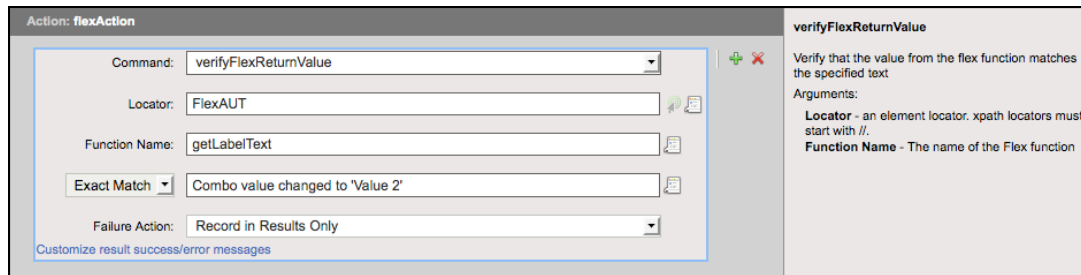
Adding Flex Validations

Once we have a Test Clip with clip elements that invoke our earlier automation work using Action Scripts, it is a simple matter to also add validations as part of our Test Clip. First, we will add another function to the action script file that allows us to get the text of the Flex label component.

```
/**
 * Get the text of the "TestFlexLabel" label
 * @return the value of the label text attribute, or null if an exception occurred
 */
private function getLabelText():String {
    var label:Label, retval:String = null, id:String="TestFlexLabel";
    try {
        label = Label(getElementById(id));
        retval = label.text;
    }
    catch(e:Error) {
    }
    return retval;
}
```

1. Next, add the ExternalInterface callback to make it available in JavaScript.

This allows us to use the **verifyFlexReturnValue** validation to verify that the label changes each time that we do one of the actions that we're testing.



```
/**
 * Register all the functions available from JavaScript
 * @return true if the functions were properly registered, false otherwise
 */
private function registerTestCallbacks():Boolean {
    var retval:Boolean = false;
    if(ExternalInterface.available==true) {
        ExternalInterface.addCallback("selectByLabel", selectByLabel);
        ExternalInterface.addCallback("clickButton", clickButton);
        ExternalInterface.addCallback("setInputText", setInputText);
        ExternalInterface.addCallback("getLabelText", getLabelText);
        retval = true;
    }
    return retval;
}
```

Adding Flex Recording Capability

To enable recording through SOASTA's browser recorder extension, a record function is added to each event handler or component function that corresponds to an action in the test. For this example, we will record changes to the comboBox, clicks on the button, and changes to the text input.

1. First, we add this function to the action script that calls out to JavaScript in the HTML page to record the event.

```
/**
 * Called by all event handlers to record the action taken that caused
 * the event. Calls a JavaScript function that in turn records the event.
 */
private function jsRecordFlexAction(id:String, functionName:String,
    param1:String, param2:String):void {
    var info:Object = new Object();
    info.id = id;
    info.functionName = functionName;
    info.param1 = param1;
    info.param2 = param2;

    if (ExternalInterface.available)
    {
        ExternalInterface.call("recordFlexAction", info);
    }
}
```

2. Next, add the recordFlexAction JavaScript to the HTML page.

```
function recordFlexAction(json)
{
    var element = document.getElementById(json.id);
    if (element != null)
    {
        element.setAttribute("recordFlexFunctionName", json.functionName);
        if (json.param1 != null)
            element.setAttribute("recordFlexParam1", json.param1);
        else
            element.removeAttribute("recordFlexParam1");
        if (json.param2 != null)
            element.setAttribute("recordFlexParam2", json.param2);
        else
            element.removeAttribute("recordFlexParam2");
        var ev = document.createEvent("Events");
        ev.initEvent("recordFlex", true, false);
        element.dispatchEvent(ev);
    }
}
```

3. Then, we add calls to jsRecordFlexAction in each of the component event handlers in the MXML file (FlexAUT.mxml).

```

<mx:Script>
  <![CDATA[
    import mx.collections.ArrayCollection;

    [Bindable]
    private var comboContent:ArrayCollection;

    private function onInit():void {
      comboContent = new ArrayCollection(['Value 1', 'Value 2', 'Value 3']);
    }

    /* Functions triggered on UI component events*/
    private function onTestFlexComboChange():void {
      this.TestFlexLabel.text = "Combo value changed to '" + this.TestFlexCombo.selectedLabel + "'";
      jsRecordFlexAction(this.id, "selectByLabel", "TestFlexCombo", this.TestFlexCombo.selectedLabel);
    }
    private function onTestFlexButtonClick():void {
      this.TestFlexLabel.text = "Button clicked";
      jsRecordFlexAction(this.id, "clickButton", "TestFlexButton", null);
    }
    private function onTestFlexInputChange():void {
      this.TestFlexLabel.text = "Text input value set to '" + this.TestFlexInput.text + "'";
      jsRecordFlexAction(this.id, "setInputText", "TestFlexInput", this.TestFlexInput.text);
    }

  ]]>
</mx:Script>

```

4. When you do so, the SOASTA recorder extension will record flexActions when the user changes the comboBox value, click on the button or types text into the text input.

SOASTA, Inc.
1975 Landings Drive
Mountain View, CA 94043
866.344.8766
<http://www.soasta.com>