

AUTOMATED UI TESTER FOR ADOBE[®] ACTIONSCRIPT[®]

USER GUIDE

Last Updated
30 May 2012

Adobe Systems Inc.

Contents

Introduction	3
How to set up Genie	3
Ensuring that Genie has been set up successfully.....	4
Various components of Genie.....	5
GenieID	5
Genie Features Overview	5
Detailed Description of Features.....	6
Genie user interface	6
<i>Genie menu</i>	6
<i>Connect to SWF drop-down menu</i>	7
<i>Working with Genie View</i>	7
<i>Genie context menu</i>	12
Different Genie status icons.....	14
Script recording and playback.....	14
<i>Set up a script project</i>	14
<i>Script recording in Genie</i>	15
<i>Script playback using Genie</i>	16
Genie Test Suite	18
Recording SWF Applications.....	19
Recording mouse actions	19
<i>Click action is performed but nothing happens</i>	19
Recording keyboard actions.....	19
<i>Command-line usage of Executor</i>	20
<i>Accessing command-line parameters passed to the Genie script</i>	21
Running Genie steps without extending the Java class from your Genie script.....	21
User-exposed methods for validation and generic use.....	22
<i>Generic component methods</i>	22
<i>Automatic Synchronization</i>	23
<i>Specific Component Methods</i>	23
<i>Script Methods</i>	24
<i>Genie Helper class</i>	24
<i>Global variables in scripts</i>	25
UI Functions	26
<i>Usage of UI functions</i>	27
Logging	28
<i>Script logs</i>	28
<i>Disabling script logging</i>	29
<i>Adding custom steps in a script log</i>	29
<i>Assertion steps in a Genie script</i>	30
Multi-SWF recording and playback.....	30
Extending GenieID at run time	30
Genie Locator	31
Capturing Flash events for validation	33
Image-based validations	33
<i>Component image shown is not the correct component image</i>	34
<i>UIImage.findImage method fails to find the image</i>	34
Dispatching events on a covering layer.....	34
Special usage scenario for <code>getValueOfDisplayObject()</code>	35
Support for a SWF application having non-English characters	35
Java Documentation for Genie.....	36

Introduction

Automated UI Tester for Adobe® ActionScript® is code named as Genie.

Automated UI Tester is a Flash/Flex automation tool that can record user actions on a SWF file and play them back with high fidelity. With other automation tools, automating Flash applications requires instrumenting them to run inside a wrapper application; but with Genie, it just requires a configuration file and Flash Player. So with Genie, users can automate their production builds easily. Genie currently works on Windows and on Mac OS X.

How to set up Genie

Follow these steps to set up the Genie environment. You can run Genie with a debug version of Flash Player:

1. Install the debug version of Flash Player (depending on your testing needs) from:
<http://www.adobe.com/support/flashplayer/downloads.html>
 Install the appropriate version of Flash Player for your operating system and web browser.
2. JDK version 1.5 is required (Recommended is JDK 1.5 Release 22) for compiling recorded Genie scripts and your other Java development needs. (You just require JRE in case you already have compiled Script classes.) The JDK can be downloaded from:
<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase5-419410.html>
3. Install the Eclipse IDE for Java Developers, version 3.5.0 or above, for writing scripts. The latest version of Eclipse can be downloaded from:
<http://www.eclipse.org/downloads/>
4. Get the latest build of Genie.zip provided to you; copy the ZIP file to any location on your disk and extract it.
Note: In subsequent pages, <GeniePath> will refer to the location where you have unzipped the Genie.zip file.
5. Copy GeniePlugin.jar from <GeniePath>\GeniePlugin\GeniePlugin.jar to the plug-in folder of your Eclipse installation folder. For example: **D:\Eclipse\Plugins**
6. Copy the mm.cfg file from <GeniePath>\GenieLibrary\mm.cfg to the user folder.
 - o C:\Documents and Settings\testuser\ (For Windows XP)
 - o C:\Users\testuser\ (For Windows 7)
 - o /Users/testuser/ (For Mac Os X)
7. Open mm.cfg and edit it to update the path to the file GenieLibrary.swf. You just need to add one line in mm.cfg, which will look like:
 PreloadSWF = <GeniePath>\GenieLibrary\GenieLibrary.swf
8. Open the "Global Security Settings panel" topic in Flash Player Help:
http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.ht

[ml](#). What looks like an image on this page is actually the working Global Security Settings panel for Flash Player in your browser.

- Select the "Always Allow" option.
- Click the "Edit locations" pop-up menu and add a location for GenieLibrary.swf. This is the same location you added to mm.cfg.
- Close the browser.

Ensuring that Genie has been set up successfully

1. Run GenieSocketServer.jar placed at <GeniePath>\GenieServer\ by typing the following command in the Windows command window or Macintosh Terminal.

```
Java -Dfile.encoding=UTF-8 -Xms512M -Xmx512M -XX:MinHeapFreeRatio=20 -
XX:MaxHeapFreeRatio=40 -jar GenieSocketServer.jar
```

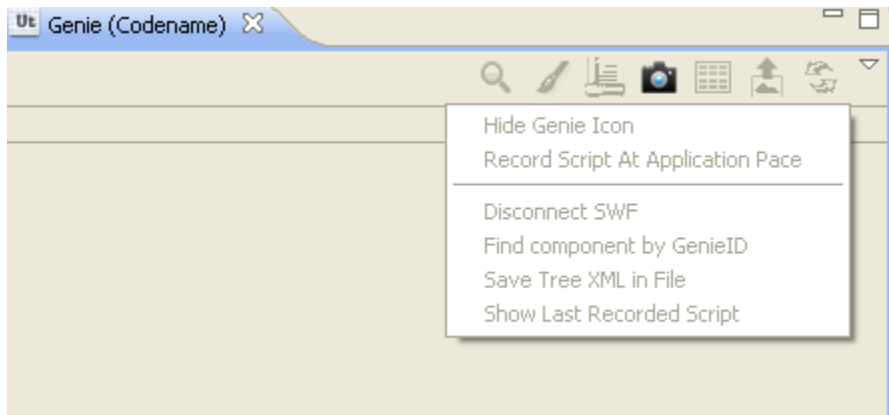
You may need to navigate to the folder where the file GenieSocketServer.jar is present, or else provide the complete path to GenieSocketServer.jar in the command above.

To run GenieSocketServer on Windows, launch the batch file LaunchServer.bat.

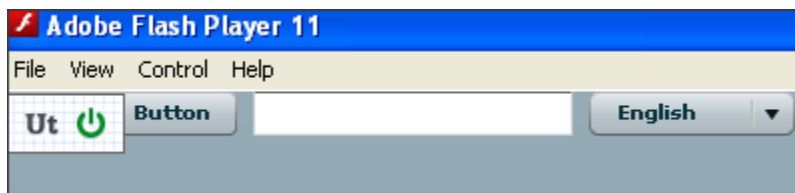
To run GenieSocketServer on the Mac, launch the shell script file LaunchServer.sh.

Note: GenieSocketServer.jar, LaunchServer.bat and LaunchServer.sh files will be available in the <GeniePath>\GenieServer Folder. You will have to edit these files to account for the location of Genie on your system.

2. Launch Eclipse.
3. To open Genie View (see below), select Windows > Show View > Other > Genie > Genie.



4. Launch any SWF file in your browser, and you should see a Genie Icon on the top left corner of the SWF file, as shown below. If you see this green icon, then you have set up Genie successfully.



The processes to record and play back a script are detailed in later sections of this Guide.

Various components of Genie

You should familiarize yourself with the following Genie components:

- **GenieLibrary.swf:** Enables recording and playback of actions on a SWF file without build instrumentation.
- **GeniePlugin.jar:** All the UI elements of Genie have been packaged in the form of an Eclipse plug-in and can be seen inside your copy of Eclipse when the Genie plug-in is installed.
- **GenieSocketServer.jar:** The Genie Socket Server works as interface/connector between Eclipse, Executor, and the SWF Library; therefore, it must always be running.
- **Executor.jar:** This Java executable is used to play back Genie scripts.

GenieID

The concept of the GenieID is pivotal for understanding the rest of this Guide. The GenieID is a unique ID assigned by Genie to each component of the Flash application (button, text, label, sprite, or other component) under test. The GenieID is the basis on which Genie recognizes each unique component, so no two components can share a GenieID. The GenieID is composed of multiple attributes of a component, and unless those attributes change, the GenieID will not change.

Genie Features Overview

Genie provides the following features to users to ease testing of Flash applications:-

1. Genie user interface (present inside Eclipse)
 - a. Genie Menu:
 - i. *Show Genie View*
 - ii. *About Genie*
 - b. Connect to SWF drop-down menu
 - c. Genie View:
 - i. *Find component in a tree*
 - ii. *Highlight Component in SWF*
 - iii. *Show local coordinates on SWF*
 - iv. *Capture Image*
 - v. *Request Genie Properties of Object*
 - vi. *Show Genie Component Image*
 - vii. *Refresh Genie Tree*
 - d. Genie context menu:

- i. *Hide Genie Icon*
 - ii. *Record Script At Application Pace*
 - iii. *Disconnect SWF*
 - iv. *Find component by GenieID*
 - v. *Save Tree XML in File*
 - vi. *Show Last Recorded Script*
- 2. Genie icon states
- 3. Script record/playback
- 4. User-exposed methods for validation and generic use
- 5. UI functions
- 6. Logging
- 7. Multi-SWF recording/playback
- 8. Extending GenieID
 - a. Genie Locator
 - b. Capturing Flex Events in Genie Script
 - c. Support for SWF having non-English characters

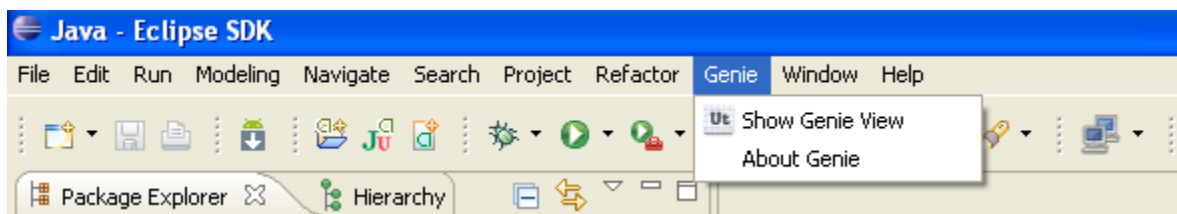
Detailed Description of Features

Genie user interface

The Genie user interface has three major subcomponents: namely, the Genie menu, the Connect to SWF drop-down menu, and Genie View.

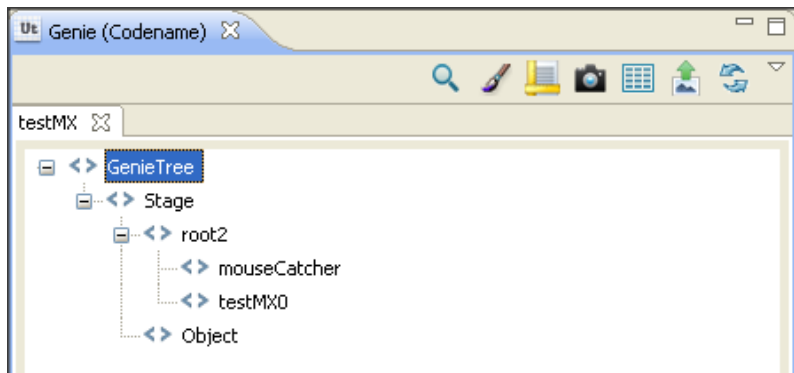
Genie menu

Once Genie is installed in Eclipse, it adds the Genie menu (shown below) to the Eclipse UI. The Genie menu enables you to open Genie View and access information about your current installation.



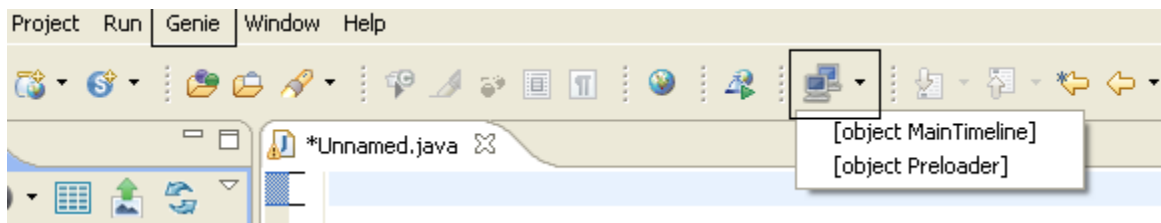
Show Genie View

This command will open a Genie View window (shown below) in which you have multiple options to interact with the SWF file that is being automated:



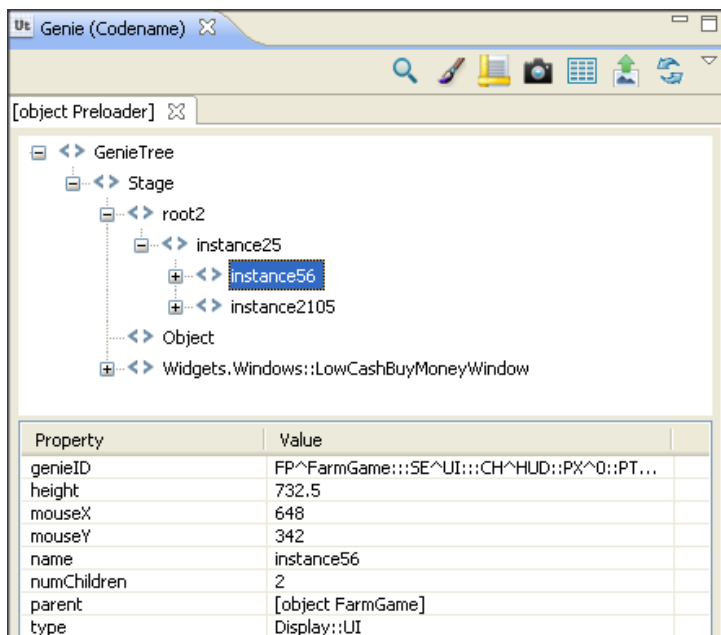
Connect to SWF drop-down menu

The Connect to SWF drop-down menu (accessed through the icon shown below) will list all the SWF files that can be automated using Genie (this includes all the SWF files active on the current machine):



Working with Genie View

Select one of the SWF files in the drop-down menu, and the object hierarchy of that SWF file will be loaded in tree mode in Genie View as shown below. For example, if you select [Object Preloader] to connect to the application SWF file (such as for the game Farmville), the following object hierarchy tree will be loaded:



In this tree, you can find the property set for each object in the SWF file, such as id, label, name, and so on. There is one unique GenieID for each SWF object. The properties table (as shown above) may not show the latest state of the application. To view the latest, refreshed properties, click "Request Genie Properties of Object" from Genie View. It shows all the properties of the selected object with their latest values.

Find component in tree

This feature searches for a component in the SWF's object tree. This helps in mapping SWF components to the tree displayed in Genie View.

To use this feature:

1. Launch and connect to a SWF application as described above.
2. Click the "Find component in tree" button in Genie View:

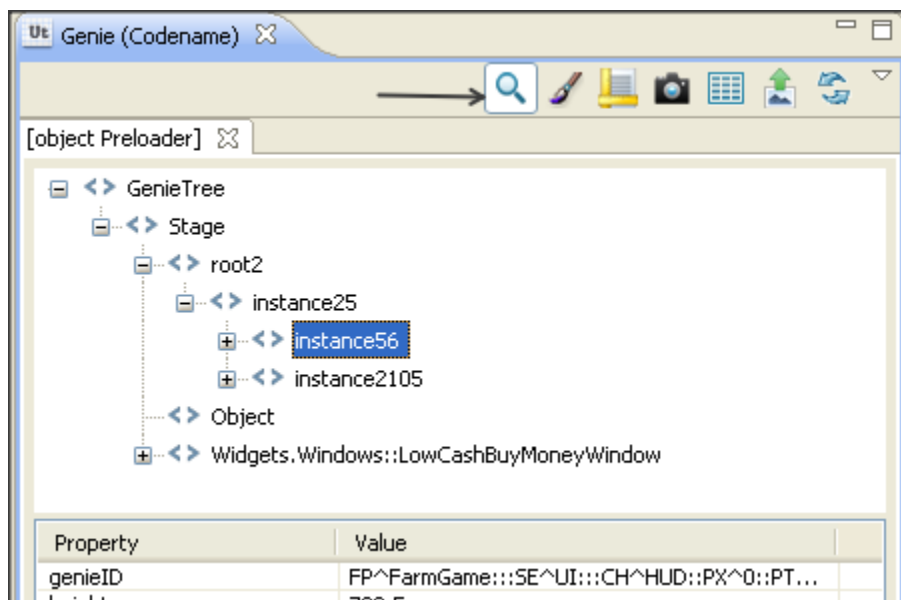
This is a pushbutton with two states:

- "Find component in tree"
- "Find component in tree and do not propagate actions"

3. Now, move mouse to the SWF file; the mouse cursor will convert to a hand.

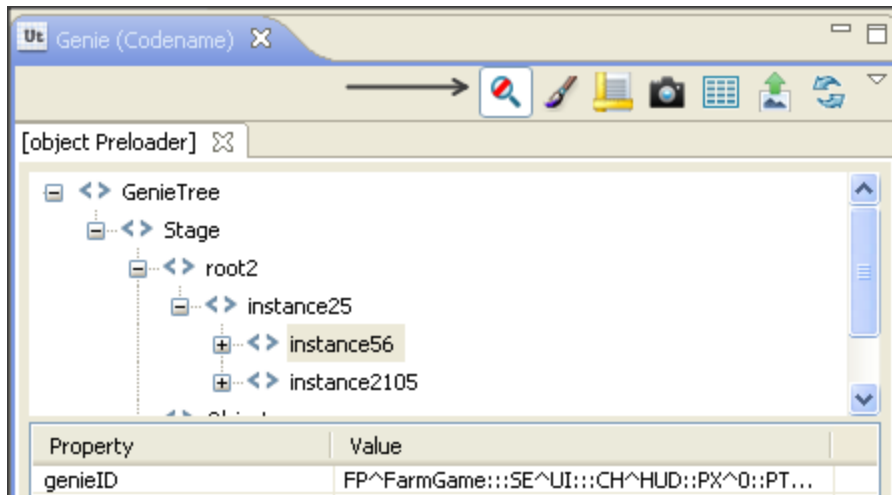
Find component in tree:

Click a SWF component in the launched SWF application, and the corresponding component in the object tree from Genie View will be selected. The click will be propagated to the SWF app and the action normally associated with that click will occur.



Find component in tree and do not propagate actions:

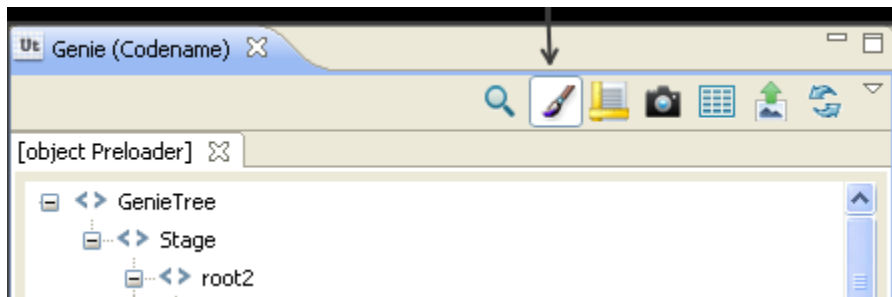
Click a SWF component in the launched SWF application, and the corresponding component in the object tree from Genie View will be selected; however, the click will not be propagated to the SWF app and no action will occur.



Highlight Component in SWF

This feature highlights a component in the SWF application that is currently being displayed in the tree hierarchy in Genie View. This action is the reverse of "Find component in tree"; so, this helps in mapping from the tree in Genie View to components in the launched SWF app.

1. Launch and connect to a SWF application as described above.
2. Click the "Highlight component in SWF" button in Genie View.

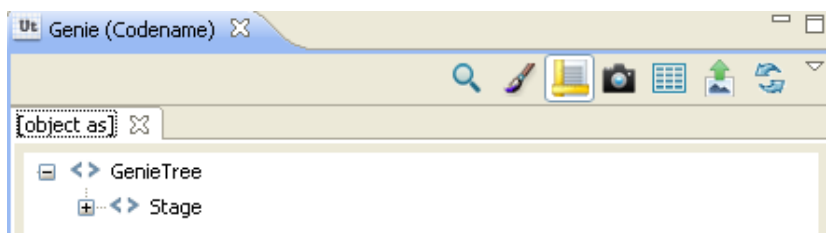


3. Now, click any object in the object tree being displayed in Genie View, and you can see the corresponding object in the SWF, highlighted in red.

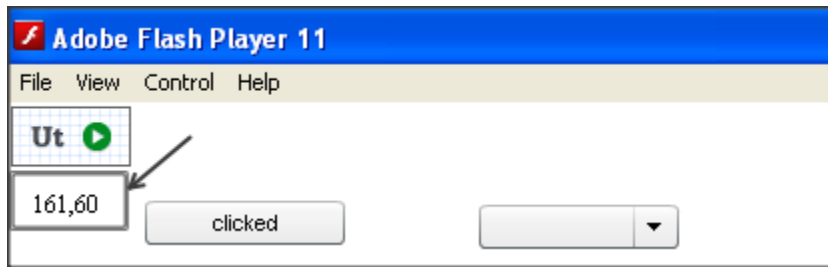


Show local SWF coordinates

To show any component's coordinates in the current SWF application, click the "Show local coordinates on SWF" icon in Genie View.



It will show the local coordinates of any component with respect to the application's Stage:

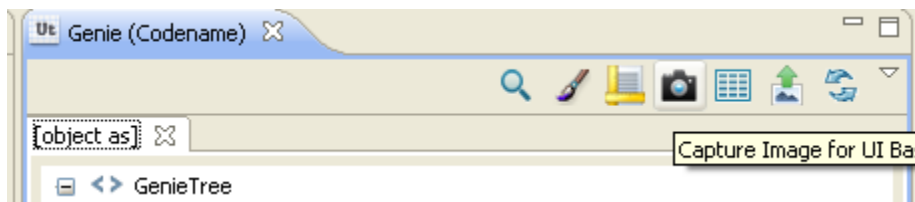


These coordinates will come in very handy while using the API from the `UILocal()` class. Please refer to the Java Docs for this class for more details (for the Java Docs, see "Java Documentation for Genie" at the end of this Guide).

Capture Image

Genie provides support to capture an image of the screen in PNG format for UI-based actions, and these captured images can be used by the image-based APIs provided by Genie.

1. Click the "Capture Image for UI Based Actions" button in Genie View:

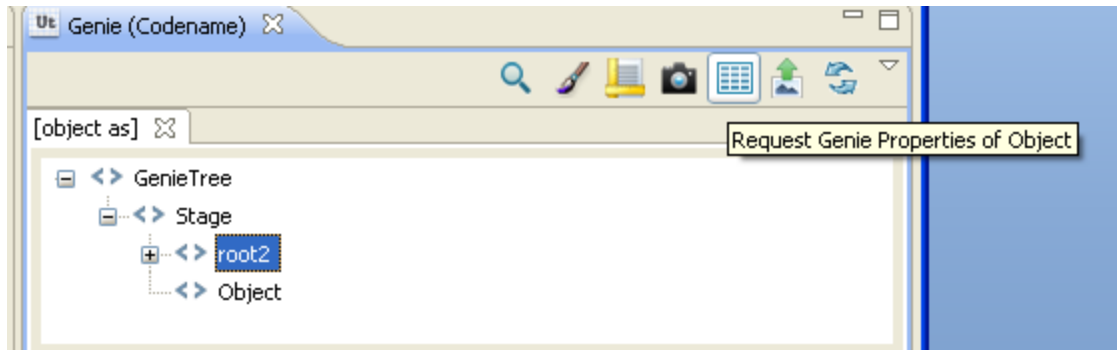


2. The Eclipse window will get minimized and the whole screen switches to capture mode (dimmed).
3. You can then capture the screen by dragging and dropping the mouse pointer anywhere on the dimmed screen.
4. When you release the mouse button, a pop-up window will appear with a preview of the captured image and the following options: Save, Recapture, Add Hot Spot, and Close.
5. You can also add a hot spot to an image, whereby you specify the point in the image on which you want Genie to click.
6. You can add clicks to these captured images in your script by using `Com.adobe.genie.executor.uiEvents.UIImage` class functions; for example:

```
new UIImage().clickImage(String imagePath , int... tolerance)
new UIImage ().clickImageHotspot (String imagePath , int... tolerance)
```
1. Pressing Esc will close the capture mode and restore the Eclipse window.

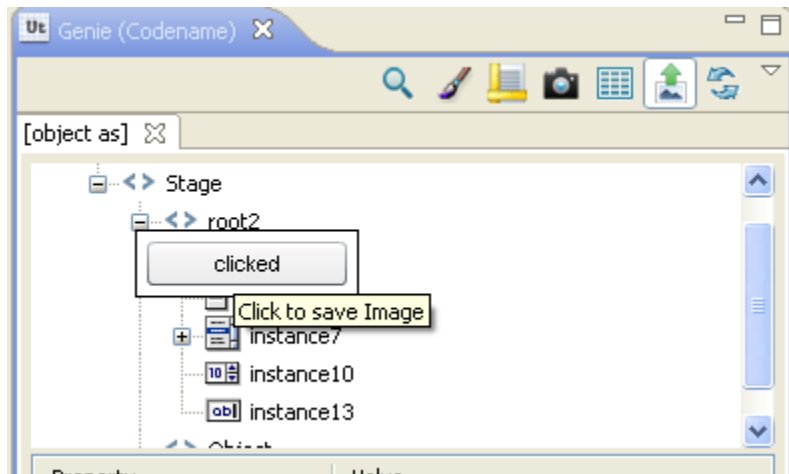
Request Genie Properties of Object

Use this option to request all the properties associated with an object in Genie View. This option fetches all the latest values for all the properties of the selected object. If the properties table in Genie View shows old properties, then use this option to fetch the latest properties at run time. This is a very handy alternative to view all the properties of a component.



Show Component Image

Switch this flag to ON, and you will be able to see images of SWF objects from the Genie View object tree just by moving your mouse over objects in the tree. Each image will be displayed at the position of mouse, and you can save that image, too, just by clicking on it. See below:



When you click the image, it will be saved in PNG format.

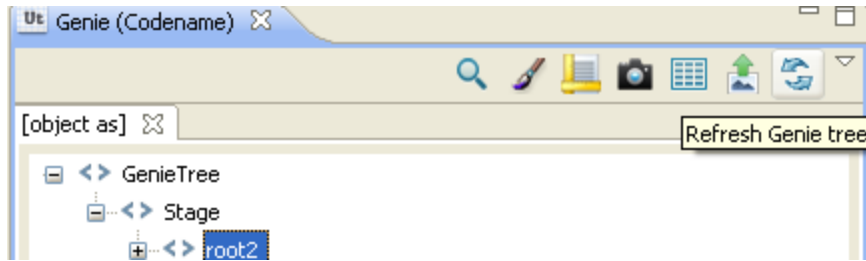
You can also capture the component image at run time using the `captureComponentImage()` method in the `GenieComponent` class. You can also capture the application image by using `captureAppImage()` method in the `GenieScript` class.

The captured PNG image can be used with the `GenieComponent.compareImage(String targetImagePath)` method, which will compare the current component image with the target image. This method can be used in validation steps where, after any step, you can compare the component image with a baseline image. This is mainly used in validating a user's moves in Flash gaming workflows. For example:

```
String imagePath = new
GenieComponent("GenieID", app1).captureComponentImage("C:\\test.png");
captureAppImage(app1, "C:\\test2.png");
Boolean matchResult = new
GenieComponent("GenieID", app1).compareComponentImage(imagePath, true);
```

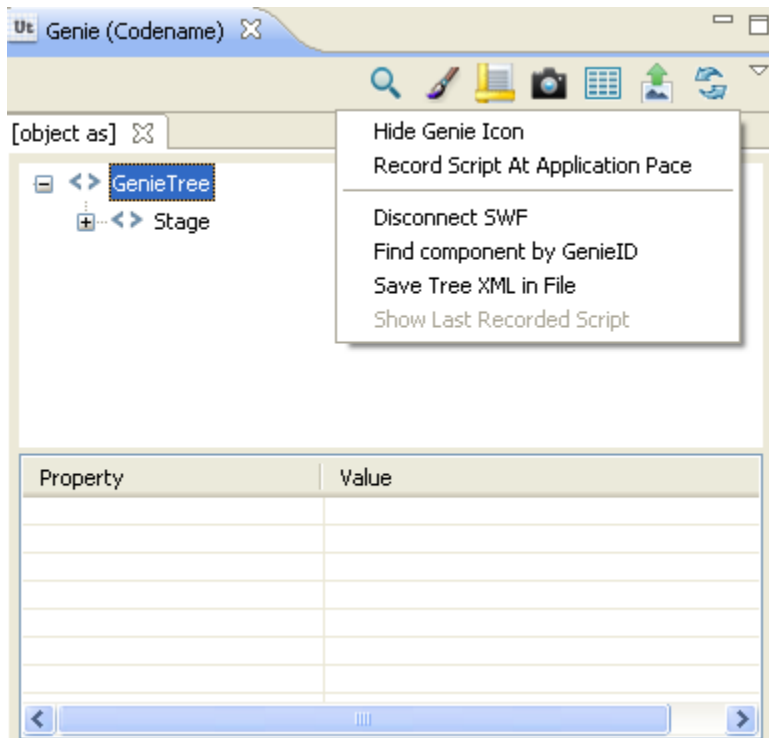
Refresh Genie Tree

This will refresh the Genie object tree and will update all property values.



Genie context menu

Now we will be covering the various options available in the Genie context menu (see below). This menu offers you options for controlling Genie's behavior and working with Genie data.



Hide Genie Icon

Choose this option to hide the Genie icon from the SWF application. This is useful when some part of the app's UI is hidden behind the Genie icon and you need to perform actions there.

Record Script At Application Pace

If you select this option before starting to record, then Genie automatically inserts `GenieComponent.waitFor(timeInSec)` statements between two consecutive statements. This way the user script gets recorded at a pace at which the application works and responds to user actions.

Disconnect SWF

Choose this option to disconnect Genie from the SWF application.

Find component by GenieID

Use this feature to search for the component that has a given GenieID in the tree view, also called the *Object Finder*.

1. Choose "Find component by GenieID" from the Genie View menu. A window with text input is shown.
2. Enter the GenieID that you want to search for in the tree and click OK. The corresponding component is shown in the Object Finder.

Save Tree XML in File

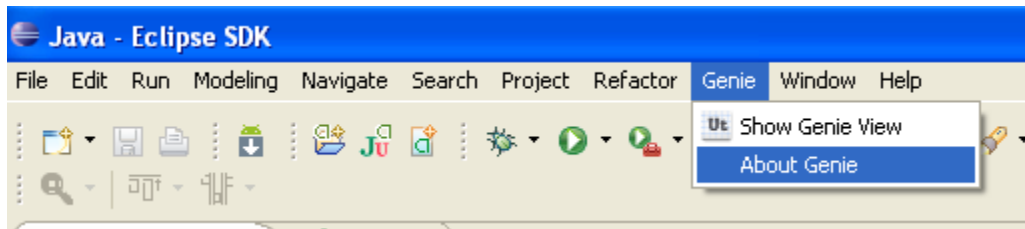
To save the tree structure shown in the Object Finder in a file, choose "Save Tree XML in File" in the Genie View context menu.

Show Last Recorded Script

To pop up a window with the contents of the last recorded script, choose "Show Last Recorded Script" in the Genie View context menu.

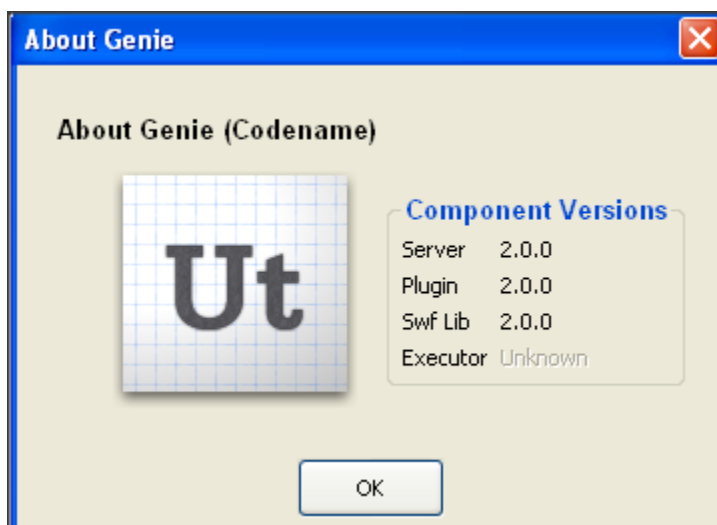
About Genie

This dialog box will give you version information about the various installed components of Genie.



Follow these steps to get the versions of all Genie components:

1. Click the Genie Menu in the Eclipse menu bar.
2. Select About Genie and the following window will be displayed.



Different Genie status icons

Following are some icons which appear on a SWF application when Genie is successfully loaded.



Disconnected: The SWF application is not yet connected to Genie Server. If this is the current image on your SWF app, you can't perform Genie operations. This means either Genie Server is not running, or a version mismatch has happened.



Connected: The SWF application has been connected to Genie Server but not with the plug-in.



Start Recording: The SWF app is connected to the Genie plug-in as well as Genie Server. Click this button to start recording.



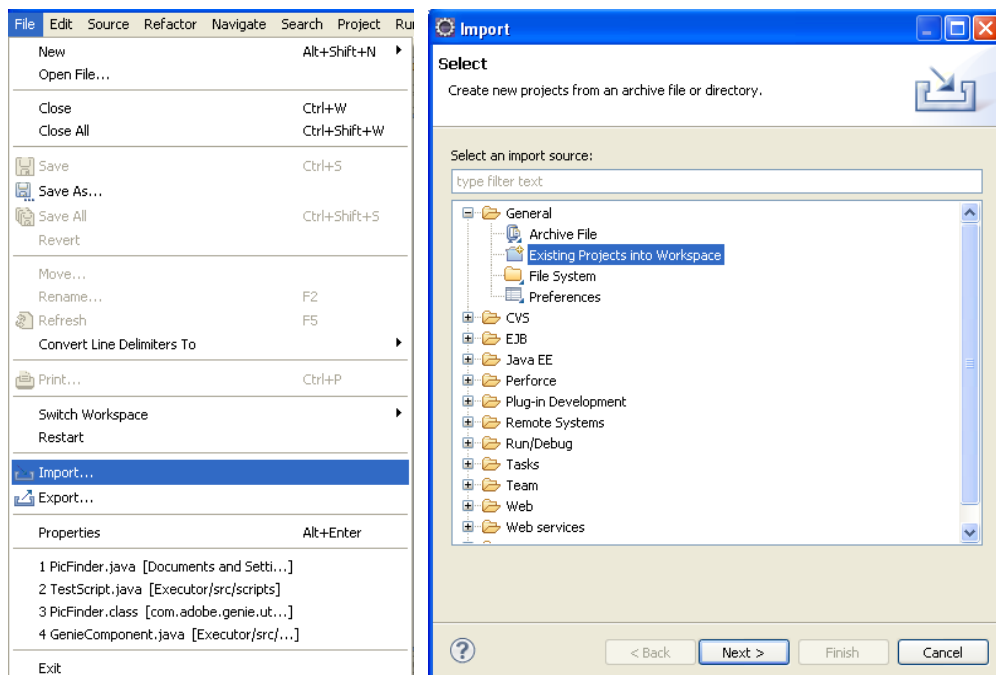
Stop Recording: You had started recording, so click this image to stop recording.

Script recording and playback

The essence of Genie is its ability to record a series of actions in a SWF application as a script for automated playback. Here's how to record and play back Genie scripts for testing your SWF apps.

Set up a script project

1. Open Eclipse.
2. Select File > Import > General > Existing Projects into Workspace as shown below:



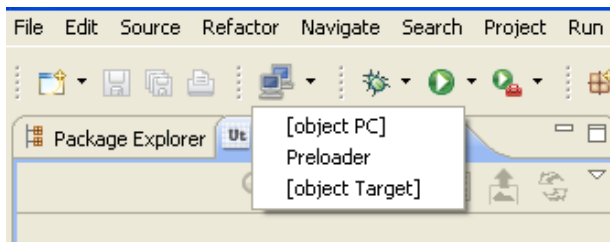
3. Select the path of the GenieScripts folder (present in the folder where the Genie.zip file has been extracted) as the root directory: <GeniePath>\GenieScripts. This will create a new project named GenieScripts in the current workspace.
4. Use this project to compile and execute your scripts using Eclipse.

Script recording in Genie

Users can record Genie scripts and play back saved scripts later. Genie records all the actions performed by the user as Flex/Flash events.

Perform the following steps to record a Script using Genie:

1. Make sure that Genie is set up and the socket server is running.
2. Launch the SWF application under test.
3. Launch Eclipse.
4. Connect to the SWF app from Eclipse using Connect to SWF from the drop-down menu:



5. The menu shows all the SWF applications currently present on Stage. Click the name of the appropriate SWF app to connect.

Note: Application names in the list can be different, depending on the type of browser; so make sure to play back the script in the browser in which it was recorded.

6. After making a successful connection, verify that the Genie icon on the top left corner of the application of the application changes to the Record button, as shown below. Click this button to begin recording:



7. Make sure that the Genie Recording button changes to a Stop Recording button (but don't click it yet!):



8. Perform whatever steps you want on the application. Genie will keep recording the steps being performed.
9. When you're ready to stop recording, click the Genie Stop Recording button. A new dialog box named Genie Script Editor will pop up, which will have all the recorded steps in a Java class format.

10. Either use the Copy Clipboard function, or just copy and paste the code to your desired location and save the script with the suitable class name; or, Copy and paste the script into the `<GeniePath>\GenieScripts\src\scripts\` folder, save the file with a suitable Java class name, and modify the constructor according to the class and file name: for example, save the file to `<GeniePath>\GenieScripts\src\scripts\abc.java`.

Note: In Java, the name of the file should be same as the name of the public class in the file.

Script playback using Genie

Before starting playback of a Genie script, make sure that Genie Socket Server is running and the application to be tested is launched and is in the same state in which the script was recorded, so that execution can proceed smoothly.

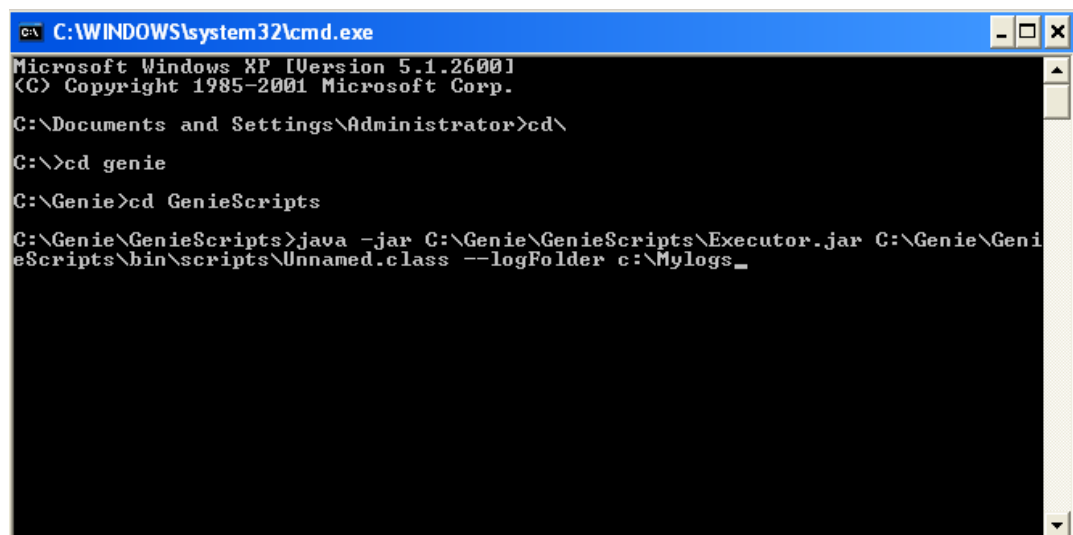
1. Compile the Genie Script generated in the recording process to a class file using either Eclipse or some other suitable tool.

Note: to compile the Genie Script file using Eclipse, just copy and paste the script file in the GenieScripts project (created in the steps under Recording, above); for example, `<GeniePath>\GenieScripts\src\scripts\abc.java`. Save the file with a suitable Java class name and use the option Project > Build all; this will compile the file. The class files will be created in the bin folder of the GenieScripts project; for example, `<GeniePath>\GenieScripts\bin\scripts\abc.class`.

2. The Genie Script can now be executed in either of two ways:
 - a. Pass the script's class file directly as parameter to the Executor.jar file on your Windows command window (shown below) or Mac Terminal. For more details of command-line options of Executor, refer to the next section.

Java -jar -Xms512M -Xmx512M -XX:MinHeapFreeRatio=20 -XX:MaxHeapFreeRatio=40
 Executor.jar *<Script's class file path>* -logFolder *<Logs folder path>*

Example: java -jar d:\Genie\executor.jar d:\Genie\Test.class -logFolder d:\Genie



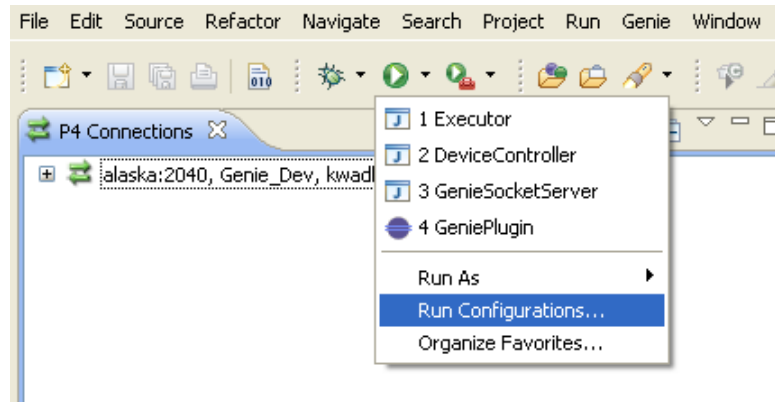
```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

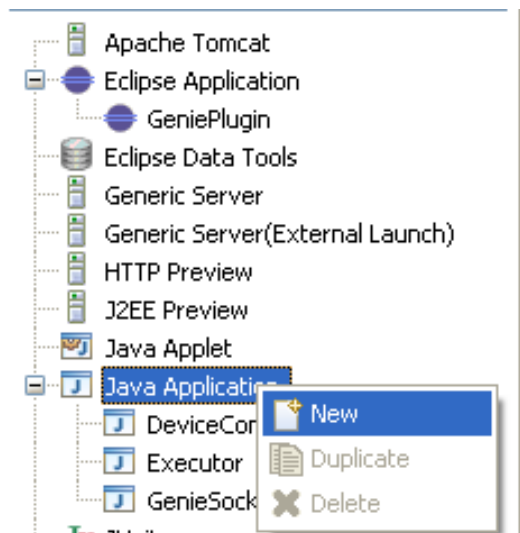
C:\Documents and Settings\Administrator>cd\
C:\>cd genie
C:\Genie>cd GenieScripts
C:\Genie\GenieScripts>java -jar C:\Genie\GenieScripts\Executor.jar C:\Genie\GenieScripts\bin\scripts\Unnamed.class --logFolder c:\Mylogs_
  
```


- b. Directly execute the script from the GenieScripts project by creating an Eclipse configuration and passing the desired script's class path as parameter. Note: to create an Eclipse configuration, follow these steps:

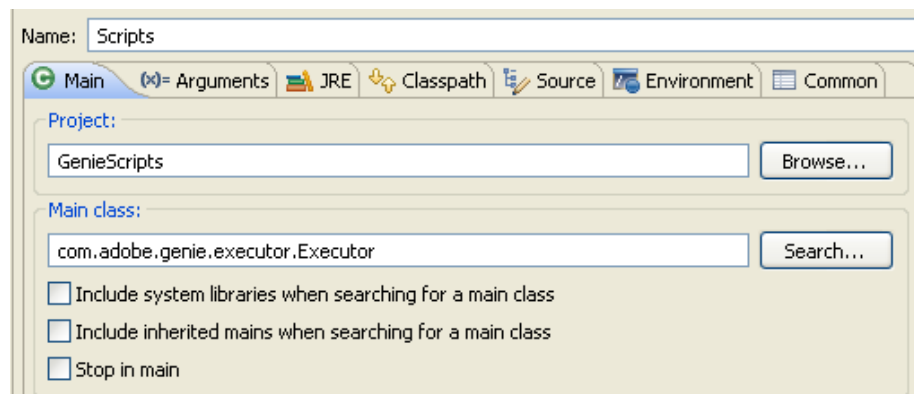
- i. Click the Run Configurations button:



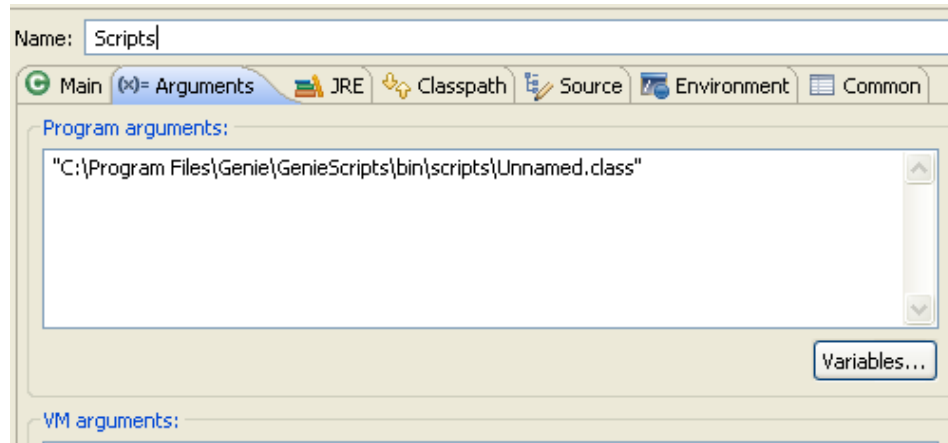
- ii. In the left pane, select Java Application > New:



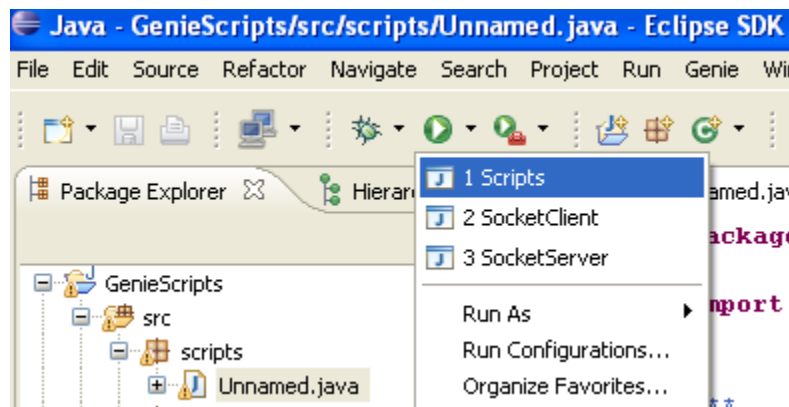
- iii. Specify GenieScripts as the project name and com.adobe.genie.executor.Executor as the Main class:



- iv. On the Arguments tab, specify the name of the Script's class file as a parameter and specify VM arguments:



- v. Eclipse will create a new Configuration named "Scripts" that user can execute directly in Eclipse.



Genie Test Suite

Test Suite is a collection of test cases that can be executed in a single command by Executor. Test Suite is represented in form of an XML file. A suite file can contain multiple suites, and any number of suites can be run by giving correct parameters on the Executor command line. See "Command Line Usage of Executor" for more details.

Example command to execute Test Suite:

```
java -jar Excutor.jar --suite suite.xml -executeSuites [InnerSuite]
```

Example suite XML:

```
<GenieSuite name="suite.xml">
  <Suite name="TestSuite" disabled="false">
    <Script disabled="false" path="C:\Genie\Test1.class" param=""/>
    <Script disabled="false" path="C:\Genie\Test2.class" param="[param1]"/>
  </Suite>
  <Suite name="InnerSuite" disabled="false">
```

```

        <Script disabled="false" path="C:\Genie\Test3.class" param="" />
    </Suite>
</GenieSuite>

```

Recording SWF Applications

Recording SWF applications entails some care and concentration to do properly, so you can fully automate your Genie scripts. Follow these instructions to set up your SWF recordings properly.

Recording mouse actions

In Flash applications, mouse actions are normally recorded with GenieSprite, GenieMovieClip, or GenieDisplayObject components. For example:

```

(new GenieDisplayObject(genieID, app1)).click(60,109,546,246,1407,594,3,false);
(new GenieSprite(genieID, app1)).click(1019,224,1019,224,1407,594,3,true);

```

The Boolean argument `false` means that this step does not require a physical mouse move and only a click event is dispatched on the component specified by the GenieID. Sometimes, however, such as in game behaviors, a physical mouse move is required before a click event is dispatched. In such a scenario, the recorded step ideally should have the Boolean argument as `true`; this will ensure that Genie first moves the mouse to the desired location, and only then dispatches a click event.

Genie records the step with the Boolean parameter as `true` **only for components** that get recorded with classes GenieSprite or GenieMovieClip. The condition for a script being recorded with the `true` parameter is that the application and the component must be in focus.

Click action is performed but nothing happens

If the recorded step does not get played back correctly, then you can manually change the last Boolean argument to `true`, which means Genie will first move the mouse to that component, and then dispatch a click event. For example:

```

(new GenieDisplayObject("root0::CN^JButton::IX^3::ITR^4", app1)).
click(102,11,957,160,0,0,3,false);
(new GenieDisplayObject("root0::CN^JButton::IX^3::ITR^4", app1)).
click(102,11,957,160,0,0,3,true);

```

This type of issue can be seen in games, depending on the game behavior.

Recording keyboard actions

Genie provides support for recording and playback of keyboard actions performed by the user. For example, here's a recording of the press of a left arrow key:

```

(new GenieDisplayObject(genieID, app1)).pressKeyForDuration("LEFT",145);

```

This step will press the left arrow key for 145 milliseconds, and then it will release the key.

Genie can record only one key at a time. This means that if, for example, the left and right arrow keys are pressed simultaneously, then the keypress detected first will get recorded with the wrong pressed duration, and the other key will get ignored. But if the key combination uses a modifier key, for example

Ctrl + right arrow, then it will get recorded correctly and will get played back as expected. The key combinations supported for this are Ctrl/Shift/Alt + any other key.

Command-line usage of Executor

Usage:

```
java -jar Executor.jar [--logFolder FolderName] [--logOverWrite] [--logFolderWithoutTimestamp] [--v] [--suite GenieSuiteXML] [--executeSuites [suite1,suite2,...]] [--noLogging] [--help] [--scriptParams [param1,param2,...]] scriptPath
```

Notes:

--scriptParams

Comma-separated list of all parameters, enclosed in square brackets [] and quotes. Example: "[a,b,c]"

--suite

Specifies the full path of the Genie Suite XML file containing the description of all suites.

--executeSuites

Comma-separated list of all suite names, enclosed in square brackets [], that need to be executed out of a full suite. If not specified, then the full suite is executed. Only valid if --suite is used.

--logFolderWithoutTimestamp

TimeStamp will not be appended to the Script log folder if the same script is executed again; the contents of the folder will be overwritten. In the case of a suite, only the final suite log will follow this rule.

--logOverWrite

No special folder will be created for storing logs and logs will be created at the root folder; also, they will be overwritten on each execution session. In the case of a suite, only the final suite log will follow this rule.

If the logFolderWithoutTimestamp option is also specified, then logFolderWithoutTimestamp will be ignored and logOverWrite will be considered.

--logFolder

Specifies a custom Log folder where the user wants the execution logs to be written.

--noLogging

Switch off script/suite logs.

--v

Shows the version of executor.

--help

Display command-line usage of executor in the console.

Accessing command-line parameters passed to the Genie script

The following example details out the way to access the `cmdArguments[]` array in the `GenieScript` class:

```
public void start() throws Exception
{
    ...
    String inputParam = "";
    if(cmdArguments.length > 0)
    {
        inputParam = cmdArguments[0];
    }
    ...
}
```

Running Genie steps without extending the Java class from your Genie script

Genie steps can also be run without extending a class from `GenieScript`. Your class can have its own entry point (for example, `main()`).

```
public class MyTest {
    /**
     * @param args
     */
    public static void main(String[] args) {
        LogConfig l = new LogConfig();

        Genie g = null;
        try{
            g = Genie.init(l);
            //connect to app
            SWFApp app1=g.connectToApp("Kuler");
            new GenieComponent(genieID, app1).waitFor(30);
        }
        catch(Exception e)
        {

        }

        finally
        {
            g.stop();
        }
    }
}
```

The `LogConfig` class contains logging options that you may want to set before creating a `Genie` object. Please see the Java Docs of this class to know more about its methods.

The Genie class does not have a public constructor, and its object can only be created by calling the `Genie.init()` method. Calling `init()` multiple times does not create different Genie objects; rather, it returns the previously created Genie object with each call. All the methods that are available when a class is extended from the `GenieScript` class are now available in the Genie class; for example, `connectToApp()`, `createAppImage()`, and others.

It is important to call the `Genie.stop()` method after you have worked with the Genie object so that socket connections can be released and clean up can be performed; so make sure to invoke this method in your `finally` block.

User-exposed methods for validation and generic use

Genie provides methods (APIs) that can be used directly in Genie scripts, either to verify certain properties and values associated with an object, or to perform some generic functions like `wait()`. All the exposed methods fall under three broad categories.

- Generic component methods
- Specific component methods
- Script methods

Generic component methods

Generic component methods are the methods exposed that are valid for all the components supported by Genie. Some of the methods that fall under this category are:

getValueOf(Property_name)

This method is used to get the value of some specific property of a component. It takes as a parameter the name of the property whose value the user wants to check, and it can be invoked on the object of any component supported by Genie. It is advisable to call the `isVisible()` function before calling this function, because if the component is not visible on Stage (the `visible` property of the component is returned as `false`), then Genie assumes that the `GenieID` of the component has changed and then takes an alternate route for obtaining the component. For example, if you want to verify the text of a text box, then you can verify this by invoking the `getValueOf()` method on the `TextBox` object and passing "text" as a parameter:

```
Boolean bVisible=false;
bVisible=(new GenieTextInput(genieID,app1)).isVisible();
If(bVisible)
    (new GenieTextInput(genieID,app1)).getValueOf("text");
```

getNumAutomatableChildren()

Gets the number of automatable children of a `GenieComponent` object.

getChildAt(index)

This method returns the immediate `GenieComponent` at the given index.

getLocalCoordinates()

Gets the local coordinates (with respect to the application and not to the screen) of the component.

isPresent()

Checks to see if the given component is available on Stage and returns a Boolean value indicating the same.

waitFor(int timeout)

This method can be used when one wants to wait for some specific component to be loaded on Stage. This method can be invoked directly using the GenieID of the component for which you need to wait, and the maximum time for which to wait. The script will keep waiting on this step till either the specified component gets loaded on Stage or the specified time is fulfilled. Example:

```
GenieComponent(genieID, app1).waitFor(10);
```

waitForPropertyValue(String property,String value,int timeInSeconds)

This method can be used when the user wants a particular property of a component to attain a specific value within the specified timeout. This method can be used to do various kinds of validations. For example, you can wait for a component to become visible before proceeding to click it. Here's another example:

```
If (new GenieComponent(genieID,app1).waitForPropertyValue("visible","false",5))
    System.out.println("pass");
else
    System.out.println("fail");
```

Many other validation methods are available that can be used to validate data after performing actions. For a complete list, please refer to Java Docs.

Automatic Synchronization

Before executing any action on a component, Genie waits for that component to appear on Stage. For example, before playing click() on a button, Genie will wait for the button to appear on Stage. It does not wait for the component to appear on the application while initializing the component.

Specific Component Methods

Certain Genie methods are valid only for specific components. Currently, some methods are exposed for Data Grid/Advanced Data Grid. The following methods fall under this category:-

getValueOfDisplayObject()

This will get value of given property for *GenieDisplayObject* control with logging. Example:

```
(new GenieDisplayObject(genieID,app1).getValueOfDisplayObject("text"))
```

getValueOfObject()

This will return value of given property for the GenieComponent control. It always returns serialized XML of the object value. Example.

```
(new GenieButton(genieID,app1)).getValueOfObject(obj);
```

Script Methods

Script methods are the methods that a user can directly invoke in their script and don't depend on any of the components. The following methods fall under this category:

setTimeout(int timeout)

This method can be used by the user to set the default timeout to be used by Genie while waiting for components.

wait(int milliseconds)

This method can be used by users to wait for a specified length of time.

getName()

Returns the name of current script.

getThread()

Returns the thread of the current script.

getTimeout()

This method gets the current timeout value (in seconds) which is currently set for actions.

isAppAvailable()

This method checks if the asked SWF app is actually launched and returns true if the SWF app is launched, else it returns false.

saveAppXml(SWFApp app, String path)

Saves Application xml in a location specified by the user.

waitForAppToConnect(String appName, int timeoutInSeconds)

This method waits for the SWF app to get launched; it waits for the specified timeout and then connects the Executor to the SWF app. It throws `ConnectionFailedException` if it cannot connect to the SWF app within the given time.

Genie Helper class

Genie has also exposed some of the commonly used APIs required to perform some generic actions such as open a URL, launch a process, and so on. These methods can be found in the class `com.adobe.genie.executor.GenieHelper`.

openURL(String url)

Opens the specified web page in the user's default browser, for example.

```
new GenieHelper().openURL("www.yahoo.com");
```

closeProcess(String processName, boolean bForce)

Close all instances of a process by name (only for Windows and Macintosh). Processes can be forcibly killed. Returns true if it succeeds.

Note: Caution should be taken while passing the process name, as all processes matching the string will be terminated.

executeSystemCommand(String[] cmd, boolean killAfterTimeout, int timeout)

Executes a system command. This is a better way of executing a command than `Runtime.getRuntime().exec()`. If `true` is passed in `killAfterTimeout`, the process is killed after the timeout happens. If this Boolean flag is set to `false`, then a timeout will be meaningless. Returns the command output as a String.

forceKillJavaProcessRunningAJar(String jarName)

Kill all Java processes that are involved in executing a particular jar (works on Windows and Macintosh). Returns `true` if it successfully killed the process.

getArrayListFromStringParams(String... args)

Returns an ArrayList of passed arguments. It takes all the strings that need to be bundled as the ArrayList as input.

captureScreenshot(String fileName)

This method captures the current screenshot and stores it in the Script log folder. It takes a String as input for the filename and stores the captured screenshot with that name in JPEG format.

captureScreenshot(String folderName, String fileName)

This method captures the current screenshot and stores it in the user supplied folder; also it takes another String as input for the filename and stores the captured screenshot with that name in JPEG format.

NOTE: A detailed description of the APIs discussed above and many more can be found in the [GenieAPISpecification.PDF](#), which has been provided along with this build. This file can be found at the location `<GeniePath>\Documentation`.

You might see some other classes in eclipse's context help which are not part of Genie API doc. These are internally used by executor project and we highly recommend not to use them as they are internal classes and might change in future.

Global variables in scripts

Genie supports following global variables. Their values change the behavior of script execution.

EXIT_ON_FAILURE:

If set to `true`, then, on encountering a step failure, Executor raises a `StepFailedException`. If this exception is not caught in the script, then Executor does not execute the rest of the steps. For example: suppose this flag set to `true`. The script has ten steps in total, and during execution, the third step fails. In this case, Executor will stop at the third step only and will not execute the remaining seven steps. The default value of this flag is `false`.

EXIT_ON_TIMEOUT:

This flag is similar to the `EXIT_ON_FAILURE` flag. If set to `true`, then, on encountering a step timeout, Executor throws a `StepTimedOut` exception. If this exception is not caught in the script, then Executor stops and does not execute the rest of the steps. The default value of this flag is `false`.

CAPTURE_SCREENSHOT_ON_FAILURE:

This flag, if turned on, will take a snapshot of the failing step (the application should be in focus for the right screen shot to get captured) and place it in the GenieLogs folder, which is present in the folder where your source script is placed and running from.

EXECUTION_DELAY_BEFORE_STEP:

This variable takes the number of milliseconds user wants to put as a delay before executing each step. The default value is zero. The user can change this between script steps.

cmdArguments Array:

This array is used to access the arguments passed to a Genie script. The user can pass command-line arguments to the script by using the `scriptParams` attribute.

Example:

```
java -jar "Executor.jar Path" "Script Class Path" -scriptParams [demo1,demo2]
```

Usage in a Genie script:

```
String firstArgument = cmdArguments[0]
```

The `firstArgument` parameter will get `demo1` as its value. It is advisable to use `cmdArguments.length` to check if parameters are passed or not.

UI Functions

If users want to perform UI operations that are not recorded using the Genie scripts, then they can manually edit the Genie scripts and embed calls to the necessary UI functions to complete their execution workflow.

UI functions come into the picture for these scenarios:

- A Flash/Flex component doesn't get recorded while recording a Genie script.
- A user wants to perform some actions directly on the screen (for example, in the case of System dialog boxes not supported by Flash Player).
- If some link opens a new tab, then the user should perform that using UI actions.

Genie provides following UI classes that can be used in a Genie script to perform UI actions:

UIGenieID()

Contains methods to perform UI actions based on a specified GenieID. These are the safest UI methods and safe even if components reflow. Example:

```
(new UIGenieID(app1)).click(genieID, 2, 2);
```

UIGlobal()

Contains methods to perform UI actions based on screen coordinates. These are purely screen coordinates-based and will result in script errors if screen resolution, platform, or other elements change.

UILocal()

Contains methods to perform UI actions based on the SWF app's local coordinates. These are dependent on Flash Player Stage and work with respect to Flash Player Stage's upper-left corner. These will work well if Flash Player changes its location or the browser window size is changed.

Example:

```
new UILocal(app1).click(100,200);
new UILocal(app1).drag(100,200,100,250);
```

UIImage()

Contains methods to perform UI actions based on some image on screen. An image should already have been captured for it to work. Capturing image functionality is built right inside Genie using an option in Genie View. Example:

```
new UIImage().clickImage("C:\\image.png", 10);
```

It will search for the image and click on the center point of the image.png file.

```
new UIImage().clickImageHotSpot("C:\\image.png", 10);
```

It will search for the image and click on the given hotspot point of image.png. The hotspot point is the point where the user wants to click on the image, and you can define the hotspot point at the time the image is captured from within the Genie UI.

UIKeyboard()

Contain methods to perform keyboard actions. Example:

```
(new UIKeyboard()).typeText("C:\\open.png");
```

This will type the text given as an argument at the current cursor location.

```
(new UIKeyboard()).typeKey(KeyEvent.VK_ENTER);
```

This will press the Enter key. User can use other keys from the KeyEvent class.

Usage of UI functions

To make use of the Genie UI functions, follow these steps:-

1. Create an object of an appropriate class from the UIEvents package (Java classes in this package have functions to perform UI operations) with the SWFApp object on which to perform UI actions.
2. Invoke the particular UI method to perform the UI action. Example:

```
UILocal uilocal=new UILocal(app1);
UILocal.click(200,200);
```

The above statement will perform a click on the coordinates "200,200" with respect to the SWF's own coordinates.

Another example:

```
SWFApp app1=connectToApp(appName);
(new UIGenieID(app1)).click(genieID,33,14);
```

This statement will click at a delta of "33, 14" with respect to the starting coordinate of the text box whose GenieID is mentioned.

Logging

All components of Genie do some logging to troubleshoot common situations. For all the components, Genie creates two types of logs:

1. Debug logs
2. Runtime logs

To create debug logs, you must launch the server in debug mode using the following command on the Windows command window or the Macintosh Terminal:

```
Java -jar GenieSocketServer.jar -debug
```

If the DEBUG flag is enabled, it will create debug logs; otherwise, runtime logs will be created for the specified component. All the logs for the different components are created at the following locations:

- Server:
User Folder > GenieLogs > GenieSocketServer > ServerLog_<TIMESTAMP>.log
- Plug-in:
User Folder > GenieLogs > GeniePlugin > PluginLog_<TIMESTAMP>.log
- Preload Logs:
User Folder > GenieLogs > PreloadLogs > Application folder > <TIMESTAMP>.log
- Executor:
 Executor Directory > GenieLogs-> <ScriptName_TIMESTAMP> folder

Script logs

For execution of every script, Genie generates a script Log, which will be created as an XML file in a folder with the current date and time as the name of the folder. If the user has not specified the log folder path as a parameter while executing the script, these logs will always be created in the directory where Executor.jar file is present: <GeniePath>\GenieScripts\Executor.jar.

Genie XML logs can be used as input to the result parser, so that Genie results can be used in an automation environment. Genie also creates logs in HTML format for viewing by the user.

The current script log structure contains:

- Product name and version
- Flash Player type, version, and SDK version
- Machine information: memory, operating system and version, and so on
- Genie version for all components
- Test script name and its consolidated time and result
- Each step's name, time taken by the step, parameters of the step, and result for that step.

By default, every step in Genie gets logged in Genie logs; however, sometimes you might wish not to log them. Genie has certain methods which take extra parameter so that logging for those methods can be disabled.

Disabling script logging

There might be few situations where you might not want to have the results of a step logged because a failure in a step will result in failure of the complete script log. So there are certain methods for which logging is optional and can be selected as on or off.

In the methods listed below, the last Boolean parameter is for logging. If the parameter is passed as `true`, then the step is logged in Genie logs; if passed as `false`, then the step is not logged in Genie logs. For the complete list, see the Java Docs for overridden methods that accept the `bLogging` flag.

Class Name: GenieScript

```
connectToApp(appName, bLogging)
```

Class Name: GenieComponent

```
getChildren(info, bLogging)
getLocalCoordinates(bLogging)
getValueOf(name, bLogging)
waitFor(timeInSeconds, bLogging)
```

Adding custom steps in a script log

If you want to add some custom steps in your script logs, you can do so by invoking the `addTestStep()` method of the `GenieScriptLogger` class in their scripts. Example:

```
GenieStepObject genieStepObject = GenieScriptLogger.addTestStep("Custom
Logging");
```

Now you can use various methods of the `GenieStepObject` Class to add parameter, message and step result properties to the Custom Step object created through `GenieScriptLogger` class.

- `addTestStepParameter`: will add a Test parameter for the step created
- `addTestStepMessage`: will add a message for the step created
- `addTestStepResult`: will add the result for the step created

Usage example:

```
public void start() throws Exception {
    SWFApp app1=connectToApp(appName);
    GenieStepObject genieStepObject = GenieScriptLogger.addTestStep("Validating if
    Standard button is pressed or not");

    if (textInputValue.startsWith("Standard Button pressed!"))
    {
        genieStepObject.addTestStepMessage("Standard Button Pressed
        Successfully",GenieLogEnums.GenieMessageType.MESSAGE_INFO);
        genieStepObject.addTestStepResult(GenieResultType.STEP_PASSED);
    }
    else
    {
```

```

        genieStepObject.addTestStepMessage("Standard Button Pressed
            failed",GenieLogEnums.GenieMessageType.MESSAGE_INFO);
        genieStepObject.addTestStepResult(GenieResultType.STEP_FAILED);
    }
}

```

Assertion steps in a Genie script

Genie scripts provide the user some assertion methods to assert equality of the actual value with the expected value. This is a convenient way of adding logging in your scripts with just one command.

Use this statement in your script to enable assertion features directly:

```
import static com.adobe.genie.executor.GenieAssertion.*;
```

Example:

```
assertEquals(String message, String expected, String actual);
```

This step will throw a `StepFailed` exception if the expected and actual strings are not equal and will add a message in the step Message only if values are different.

```
assertNotNull(String message, Object object);
```

This step will verify that object is not null. If it is null, then it will throw a `StepFailed` exception.

If `EXIT_ON_FAILURE = true` flag is set, then the script will not proceed after failure of the assertion step.

All assertion steps have logging enabled with `GenieStepType.STEP_ASSERTION_TYPE` and will have expected and actual values in Genie logs also.

Multi-SWF recording and playback

Genie provides support to record and play back scripts on multiple SWF apps simultaneously.

For a multi-SWF workflow, follow these steps:

- While recording a script, you need to start recording on each SWF app manually.
- While stopping recording, make sure that it has been stopped from all the Swfs

Note: For a multi-SWF scenario, Genie will consider the recording to be complete when it has been stopped from all the SWF apps, and will generate a single script only after stopping script recording from all SWF apps.

Also, while playing back in a multi-SWF scenario, make sure that all SWF apps needed for the playback are launched, either previously or from within script, as the script will terminate if it is not able to connect to even one of the SWF apps in question. If some UI event has to be performed for a SWF app, make sure during playback that the SWF in question is visible on screen.

Extending GenieID at run time

Genie assigns a unique ID to every automatable component in the application. But there can be situations where a GenieID changes, primarily when a component is dynamic and underlying components are not being disposed properly. Genie provides a way wherein users can also specify properties in GenieID, and Genie considers those properties while playing back the object. Specifying a property ensures that the

action is performed on the correct component that has those attributes as well, which are specified by user.

Syntax:

```
OriginalGenieID$propertyName1=value1,propertyName2=value2
```

Example:

```
GenieComponent gc = (new
GenieButton("genieID$label=MyButton,className=Button", app1));
gc.click();
```

Here, the GenieID has been extended to specify the properties label and classname. While playing back the click operation, Genie will also consider these properties and play back only if the component's label and classname match the given values.

Consider these points when specifying properties:

- The functionality looks recursively in the hierarchy of the given GenieID for the specified key-value pair and, if found, will perform the action on the GenieID **provided by the user**.
- This functionality is only valid for performing playback actions like click, type, and so on.
- This functionality does not hold true for functions like `getValueOf()`. If given, genie will split based on \$, figure out the GenieID and will get the value from that GenieID.

Genie Locator

The Genie Locator feature is a programmatic way of finding a component from within the script using some of that component's attributes. For example, you may want to find all the checkboxes present on Stage which are enabled; or you may want to find a component with a particular class name, or some other such criteria. This feature is extremely useful for doing validations of the steps performed by a user and in situations where the GenieID of a dynamic component is not stable and is changing.

Currently there are two functions by which the component can be fetched:

getChildren()

- Parameters: The GenieLocatorInfo object and the bRecursive flag (Boolean)
- If the bRecursive flag is false, then it traverses the direct children of the object with the specified GenieID.
- If the bRecursive flag is true, then it recursively traverses the children of object with the specified GenieID.
- If the GenieID of the GenieComponent is blank, then the bRecursive flag should be true. With these parameters it traverses the whole application recursively to find the component.

getParent()

- It returns the Genie component associated with the specified GenieID.

By default, GenieLocator provides the following properties:

1. **Id:** the ID of the component as provided by the developer

2. **className**: String value after the ::; for example, Button in mx.controls::Button
3. **qualifiedClassName**: the full className; for example, mx.controls::Button
4. **label**: the label of the component (if present)
5. **enabled**: the enabled property of component
6. **index**: the index of the child in its immediate parent
7. **text**: the text property of the component
8. **visible**: the visible property of component
9. **propertyValueTable**: a hashtable to specify properties other than those mentioned above.

If you want to specify an additional property, it can be done by filling in a hashtable with the property on the basis of which you want to perform the search.

Example:

```
GenieLocatorInfo info =new GenieLocatorInfo();
Hashtable<String,String> myhash =new Hashtable<String, String>();
myhash.put("width","1600");
GenieComponent[] comps = (new GenieComponent(genieID, app1)).getChildren(info,
true);
```

```
//Returns all children which have a property width==1600
```

Another example:

```
GenieLocatorInfo info = new GenieLocatorInfo(); //creates the GenieLocatorInfo
Object
info.className = "LinkButton"; //Specify the property that needs to be searched

GenieComponent[] comps = (new GenieComponent("", app1)).getChildren(info, true);
//It will traverse through the whole application and will return the array of
Genie Components that matches the criterion

GenieComponent comps1 = comps[0].getParent().getParent(); //gets the parent's
parent of the component.
info.label = "My Cases"; //replaces the params in the info object

comps = (new GenieComponent(comps1.getGenieID(), app1)).getChildren(info, false);
//Will traverse the immediate children of the parent whose genieID matches with
the given genieID.
```

If value of a property is not fully known, then the wildcard character asterisk (*) can be used while specifying values in GenieLocatorInfo.

```
info.label = "Text*";
```

It will fetch objects with any label starting with "Text". The asterisk is not applicable for index and qualifiedClassName properties.

Capturing Flash events for validation

Flash events dispatched by the SWF application can be captured, and the event information can be retrieved in a Genie script. The process to do so is detailed out in this section.

To capture an event, use the `attachEventListener()` function on `GenieComponent`. Example:

```
GenieComponent comp = new GenieComponent("GenieID", app1)
String eventID = comp.attachEventLisenter("click");
```

It takes the event name as argument and returns a GUID.

To capture event details, use the `waitForEvent()` method. It takes a GUID returned by `attachEventListener()` and a timeout value. This method waits for the event to happen and returns event information in the form of a hashtable. Example.

```
comp.waitForEvent(eventID, 10);
```

If the given event is not dispatched, then `waitForEvent()` times out and the step fails.

Example script:

```
GenieComponent comp = new GenieComponent(genieID, app1)
String eventID = comp.attachEventLisenter("click");

//causes component to click and dispatch "click" event which we are listening
comp.click();
Hashtable<String, String> table = comp.waitForEvent(eventID, 10);

//Iterate hash table to access values returned by waitForEvent
Set<String> set = table.keySet();
Iterator<String> itr = set.iterator();
while (itr.hasNext()) {
    String key = itr.next();
    System.out.println(key + ": " + table.get(key));
}
```

Image-based validations

Genie provides a functionality wherein an image of each component can be captured and can be used for validation purposes.

To capture a component image, enable "Show Genie Component Image" mode in Genie View and move the mouse over a component node in the Genie tree. Moving the mouse shows the component image. Click this image to save it in PNG format on the disk. This saved image can then be passed to the validation API.

```
GenieComponent(genieid, app1).compareImage(String targetImagePath)
```

This method captures the image of the given component, compares it with the `targetImagePath` provided, and returns a Boolean value.

A component image can also be captured programmatically using the `captureComponentImage()` method of the `GenieComponent` class.

A full application image can also be captured using the following API:

```
captureAppImage(SWFApp app,String imagePath);
```

For comparison, you can use Java methods to compare application images.

Component image shown is not the correct component image

If you are trying to figure out the component by enabling the select/highlight feature of Genie and the component selected in the Genie tree does not show the correct component image and it might be the parent of the actual component holding the correct image, sometimes you might have to drill down the hierarchy farther to reach the actual component for validation.

You can also use the “Show Images with Hierarchy” feature of the Genie plug-in. On right-click of any component in the Genie tree, you can see the visual image of the selected component and its complete child hierarchy. Then, clicking on any image in that window will take you to the actual component in the Genie tree so you can get the correct component for validation.

UIImage.findImage method fails to find the image

The `UIImage.findImage()` method will fail to find the image captured from `captureComponentImage()`, as this method will capture the image of the component only, and the image background will remain white irrespective of what the actual background is in the SWF app, while the `UIImage.findImage()` method will try to find the same image with matching color pixels, including the background pixels from the SWF app as well. Thus, the image captured by `GenieComponent.captureComponentImage()` can be used with the `GenieComponent.compareImage(<path of image>)` method only.

Dispatching events on a covering layer

In the world of Flash, there are instances where an event is listened by a sprite layer or any other layer which encapsulates the object underneath it. The object might be placed and visible on the screen with respect to Stage; thus, enabling `showLocalCoordinates` will show the Stage coordinates of the object, and dispatching the click event on the same coordinates on the layer might not actually click the object in question because the coordinates might be different with respect to the layer.

Hence, this functionality is provided to get the coordinates with respect to the layer, which may or may not be equal to the Stage coordinates, depending on the app architecture. For instance, in games in which the layer that listens to the event is centrally aligned to the Stage, due to which the Stage coordinates are different from local coordinates of the layer, dispatching the event on Stage coordinates will not actually click the object.

```
GenieLocatorInfo info = new GenieLocatorInfo();
info.propertyValueTable.put("name", "value");
GenieDisplayObject[] obj = new GenieDisplayObject(genieID,
app1).getChildren(info, true);
genieID = genieID of the component containing the object having name = value
```

```
Point p = new GenieDisplayObject(obj[0].getGenieID(),
app1).getRelativeCoordinates(genieID);
genieID = genieID of the layer that listens to the events.
(new GenieSprite(genieID,app1)).click(p.x+20, p.y+15,p.x+10,
p.y+10,760,620,3,true);
genieID = genieID of the layer that listens to the events.
```

Special usage scenario for `getValueOfDisplayObject()`

This method can be used to fetch the value of the property of a component by providing just the name of the property. Explained below is the scenario in which, if you want to validate that the image has changed after performing certain actions, then there are two scenarios using which validation can be done:

If we want to fetch the value of the component that changes the image from, say, `sampleImage1` to `sampleImage2` by performing a click, then the same can be validated by using properties that contain the image type. Example:

```
SWFApp app1=connectToApp(appName)
new GenieComponent(genieID,app1). getValueOfDisplayObject ("type");

returned: Classes::sampleImage2
```

If the image type is in `flash.display::Bitmap`, then pass the `bitmapData` property to the `getValueOfDisplayObject()` method, and it will return the name of the image in the form of an object:

```
SWFApp app1=connectToApp(appName)
new GenieComponent(genieID,app1). getValueOfDisplayObject ("bitmapData");

returned: [sampleImage2]
```

Support for a SWF application having non-English characters

Genie supports Japanese/UTF-8 encoding characters, but to get the right communication to happen between all components of Genie, all of them should be started first.

Start Genie Socket Server as:

```
java -Dfile.encoding=UTF-8 -Xms512M -Xmx512M -XX:MinHeapFreeRatio=20 -
XX:MaxHeapFreeRatio=40 -jar GenieSocketServer.jar
```

To save Japanese, Chinese, or any other language characters in the script's java file, you need to adjust a setting in Eclipse. By default, the java file type is set to default encoding. You need to set it to the UTF-8 encoding scheme:

1. Go to: Eclipse menu > Window > Preferences > General > Content Types.
2. On the right side, look for "Java Source File". Set its default encoding to "UTF-8" and click Update.
3. Refresh your Eclipse project once.

To enable Executor to understand Encoding characters:

1. Open Eclipse.

2. Open Debug Configuration or Run Configuration.
3. Select Executor project.
4. Click the Common tab.
5. Change "Console Encoding" to UTF-8.
6. Click Apply and then Run.

To run from the command line:

```
java -Dfile.encoding=UTF-8 -jar Executor.jar <Script class file>
```

Java Documentation for Genie

Comprehensive Java documentation for Genie can be found at

<GeniePath>\GenieScripts\doc\GenieDoc.zip. Extract the zip file and open

<GeniePath>\GenieScripts\doc\GenieDocs\index.html.