Plugin Beanshell

Mikaël Marche

April 26, 2006

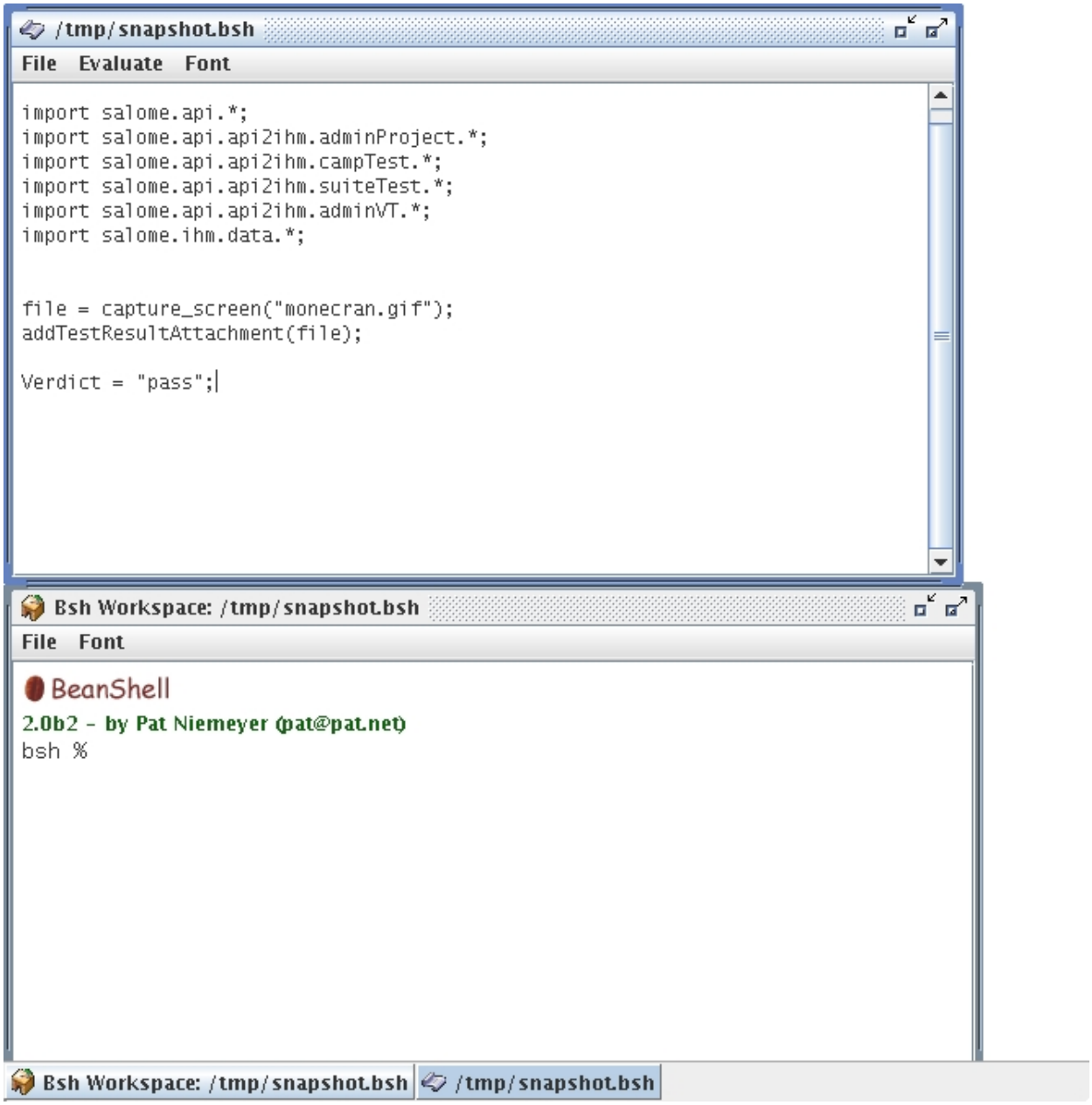
Beanshell, is a script language, fully integrated with Java, which can execute compiled Java objects or scripts written in Beanshell. Moreover, like any script and test plugin, Beanshell scripts can use references to objects present in Salomé projects.

Contents

1 Creating a Beanshell test or script

For creating a BeanShell script in environments and executions, you have to select a ń .bsh ż file in the file selection window. For creating a BeanShell script after having created an automatic test of type BeanShellTester, you have to add a *.bsh* script, starting from the *Script* tab of the test.

Any BeanShell script is an executable program which can be edited in the « Script » menu of tests, environments and executions.



2 Using the Salomé API inside BeanShell

By default, BeanShell scripts have references on the following variables (*=org.objectweb.salome_tmf.data):

Name	Class	Description	Available
date	*Java.lang.Date*	Execution date	T, Env, Ex
time	*Java.lang.Time*	Execution hour	T, Env, Ex
salome_projectName	*Java.lang.String*	Current project name	T, Env, Ex
salome_projectObject	**.Project*	Salomé project object reference	T, Env, Ex
salome_debug	*boolean*	False when the script is evaluated inside the editor	T, Env, Ex
salome_ExecResultObject	**.ExecutionResult*	Reference on the results of the current execution	T, Ex
salome_ExecTestResultObject	**.ExecutionTestResult*	Reference on the current test execution result	T
salome_CampagneName	*Java.lang.String*	Current campaign name	T, Env, Ex
salome_CampagneObject	**.Campaign*	Reference on the current campaign	T, Env, Ex
salome_environmentName	*Java.lang.String*	Current environment name	T, Env, Ex
salome_environmentObject	**.Environment*	Reference on the current environment	T, Env, Ex
salome_ExecName	*Java.lang.String*	Current execution name	T, Env, Ex
salome_ExecObject	**.Execution*	Reference on the current execution	T, Env, Ex
salome_TestName	*Java.lang.String*	Current test name	T
salome_TestObject	**.Test*	Reference on the current test	T
salome_SuiteTestName	*Java.lang.String*	Current test suite name	T
salome_SuiteTestObject	**.TestList*	Reference on the current test suite	T
salome_FamilyName	*Java.lang.String*	Current family name	T
salome_FamilyObject	**.Family*	Reference on the current family	T
testLog	*Java.lang.String*	Test log that will be added as attachment to the execution	T
Verdict	*Java.lang.String*	Test verdict (pass, fail, or unconclusive)	T
salome_classloader	*Java.net.URLClassLoader*	Class loader for BeanShell	T, Env, Ex

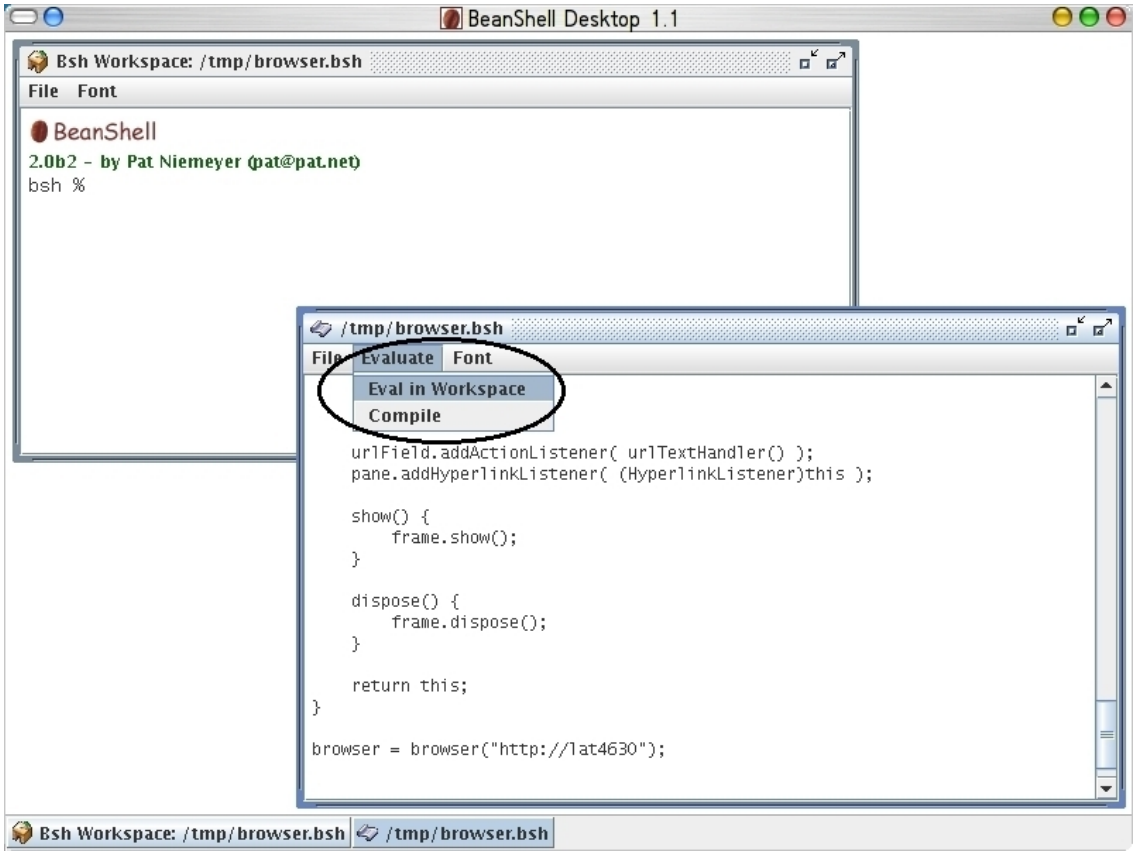
The « Available » column describes where those variables can be used (T for inside tests, Env for inside environments, and Ex for inside Executions).

Those variables are not the only ones that can be accessed inside scripts. Variables (and their values) that are defined in a data set or an environment can also be accessed inside scripts, directly through their names, and they are given the type *Java.lang.String*.

A Hashtable named « testargs », that contains the whole set of predefined variables and test variables, is accessible inside BeanShell test scripts.

3 Compiling and evaluating a script

When modifying a script, it is possible to evaluate and compile a script.



When evaluating a script, the variables which are directly related with a campaign execution are not valued (dataset values), and the variable *salome_debug* is set to *true*.

4 Predefined BeanShell functions

The file *salome_testcmd.bsh* in the *commands* directory of the BeanShell plugin countains a set of BeanShell functions which can be used in the whole set of test scripts, environments and executions.

```
import org.objectweb.salome_tmf.api.*;
import org.objectweb.salome_tmf.api.api2ihm.adminProject.*;
import org.objectweb.salome_tmf.api.api2ihm.campTest.*;
import org.objectweb.salome_tmf.api.api2ihm.suiteTest.*;
import org.objectweb.salome_tmf.api.api2ihm.adminVT.*;
import org.objectweb.salome_tmf.data.*;
import salomeTMF_plug.beanshell.ParentClassLoader;

import java.awt.image.BufferedImage;
import java.io.File;
import java.net.URL;
```

```
/* Adding a path (String or URL) to the class loader */
addJar( path ) {
        URL url;
        if ( path instanceof URL ) {
                url = path;
        } else {
            try  {
                url = new URL(path);
            } catch (e){
                url = pathToFile( path ).toURL();
            }
        }
        salome_classloader.AddJar( url );
}

/* Waiting a time t */
waittime(int t){
    Thread.sleep(t);
}

/* Fetching a copy of the screen and saving it in out_file (File) */
File capture_screen(String out_file) {
    BufferedImage image;
    Thread.sleep(5);
    Toolkit tk = Toolkit.getDefaultToolkit();
    tk.sync();
    Rectangle screen = new Rectangle(tk.getScreenSize());
    Robot robot = new Robot();
    robot.setAutoDelay(0);
    robot.setAutoWaitForIdle(false);
    image = robot.createScreenCapture(screen);
    file = new File(out_file);
    javax.imageio.ImageIO.write(image, "png", file);
    return file;
}

/* Adding a file as an attachment to the execution result of the current test */
addTestResultAttachment (File f){
    if (salome_debug == false) {
        salome_ExecTestResultObject.addAttchment(f);
    }
}

/* Adding a file as an attachment to the execution of the current campaign */
addExecResultAttachment (File f){
    if (salome_debug == false) {
        salome_ExecResultObject.addAttchment(f);
    }
}

/* Adding a file as an attachment to the current test object */
addTestAttachment(File f){
    if (salome_debug == false) {
        salome_TestObject.addAttchment(f);
    }
}

/* Adding a file as an attachment to the current test suite */
```

```
addSuiteAttachment(File f){
    if (salome_debug == false) {
        salome_SuiteTestObject.addAttchment(f);
    }
}


addTestResultAttachmentUrl (URL u){
    if (salome_debug == false) {
        salome_ExecTestResultObject.addAttchment(u);
    }
}

addExecResultAttachmentUrl (URL u){
    if (salome_debug == false) {
        salome_ExecResultObject.addAttchment(u);
    }
}

addTestAttachmentUrl(URL u){
    if (salome_debug == false) {
        salome_TestObject.addAttchment(u);
    }
}

addSuiteAttachmentUrl(URL u){
    if (salome_debug == false) {
        salome_SuiteTestObject.addAttchment(u);
    }
}

addEnvAttachmentUrl(URL u){
    if (salome_debug == false) {
        salome_environmentObject.addAttchment(u);
    }
}

addTestResultAttachmentText (String fileName, String text){
    if (salome_debug == false) {
        salome_ExecTestResultObject.addAttchment(fileName, text);
    }
}

addExecResultAttachmentText (String fileName, String text){
    if (salome_debug == false) {
        salome_ExecResultObject.addAttchment(fileName, text);
    }
}

addTestAttachmentText(String fileName, String text){
    if (salome_debug == false) {
        salome_TestObject.addAttchment(fileName, text);
    }
}

addSuiteAttachmentText(String fileName, String text){
    if (salome_debug == false) {
        salome_SuiteTestObject.addAttchment(fileName, text);
```

```
    }
}

addEnvAttachmentText(String fileName, String text){
    if (salome_debug == false) {
        salome_environmentObject.addAttchment(fileName, text);
    }
}
```

This file can be completed by the Salomé administrator.

5 Example

The following example introduces how to use the BeanShell plugin for environment, execution and test scripts. A feature of this example is to show that the execution context is kept when executing a campaign. Moreover, it shows how to use test variables and predefined functions.

The example is made of four scripts:

- An environment script declaring a web browser window, linked with an environment declaring a variable *url_test*.
- A pre-execution script printing the declared browser window in the environment.
- A test script using the environment browser and the *url_test* variable.
- A post-execution script closing the environment browser window.

The environment script is the following one:

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import java.awt.event.*;
import java.awt.*;

browser( startingUrl ) {

        invoke( method, args ) {}

        windowClosing(WindowEvent we) {
                we.getWindow().setVisible(false);
        }

        hyperlinkUpdate( HyperlinkEvent he ) {
                type = he.getEventType();
                if (type == HyperlinkEvent.EventType.ENTERED) {
                        pane.setCursor(
                                Cursor.getPredefinedCursor( Cursor.HAND_CURSOR) );
                        statusBar.setText(he.getURL().toString());
                } else
                        if (type == HyperlinkEvent.EventType.EXITED) {
                                pane.setCursor( Cursor.getDefaultCursor() );
                                statusBar.setText(" ");
                        } else {
                                setPage( he.getURL() );
                                if (urlField != null)
                                        urlField.setText(he.getURL().toString());
```

```
                }
        }

        // This is the equivalent of an inner class in bsh.
        urlTextHandler() {
                actionPerformed(ActionEvent ae) {
                setPage( ae.getActionCommand() );
                }
                return this;
        }

        setPage( url ) {
                try {
                pane.setPage( url );
                } catch(Exception e) {
                        statusBar.setText("Error opening page: "+url);
                }
        }

        show() {
                frame.show();
        }

        dispose() {
                frame.dispose();
        }

        frame = new JFrame("Browser");
        frame.setSize(800,600);
        frame.addWindowListener( this );
        urlPanel = new JPanel();
        urlPanel.setLayout(new BorderLayout());
        urlField = new JTextField(startingUrl);
        urlPanel.add(new JLabel("Site: "), BorderLayout.WEST);
        urlPanel.add(urlField, BorderLayout.CENTER);
        statusBar = new JLabel(" ");
        pane = new JEditorPane();
        pane.setEditable(false);
        setPage( startingUrl );
        jsp = new JScrollPane(pane);
        frame.getContentPane().add(jsp, BorderLayout.CENTER);
        frame.getContentPane().add(urlPanel, BorderLayout.SOUTH);
        frame.getContentPane().add(statusBar, BorderLayout.NORTH);

        urlField.addActionListener( urlTextHandler() );
        pane.addHyperlinkListener( (HyperlinkListener)this );
        return this;
        }

browser = browser("http://www.beanshell.org");
```

We can note the definition of methods *setPage, show* and *dispose* which will be used respectively in the test script and the pre- and post- execution scripts.

The pre-execution script is the following one:

```
browser.show();
```

The post-execution script is the following one:

```
browser.dispose();
```

At last, the test script is the following one:

```
waittime(5000);

/* Fetching the current screen */
file = capture_screen("before.gif");

/* Adding the fetch as attachment to the current test execution result */
addTestResultAttachment(file);

/* Asking the browser to print the URL valued by the environment parameter "url_test" */
browser.setPage(url_test);

waittime(5000);

/* Fetching the current screen */
file = capture_screen("after.gif");

/* Adding the fetch as attachment to the current test execution result */
addTestResultAttachment(file);

/* Set the test verdict to PASS */
Verdict = "pass";

/* Adding an attachment with OK in the test execution result */
testLog = "OK";
```

The test execution result has got three attached files: the two fetched screens *before.gif* and *after.gif*, and the test log *log_exec_Goto.txt* with *OK* in it.

When executing with *url_test = http://www.pagesjaunes.fr* as value for the environment parameter, the fetched screens are the following: