JxBrowser
Programmer's Guide

Version 1.3
July 10, 2009

TeamDev

Table of Contents

Chapter 1. Introduction

All Java programmers who have ever developed a Web site testing tool or desktop Java application with an HTML viewer needed a library that could load, display an HTML document and access to its content. Of course, the implementation of such a library from scratch may take a lot of programming and testing efforts. Therefore, integrating the existing browser is a preferable way. JxBrowser is a cross-platform library that enables seamless integration of Mozilla Firefox web browser into Java AWT/Swing applications on Windows, Linux, Mac OS X platforms. The library uses the Gecko layout engine for rendering HTML documents, thus ensuring compliance with many Internet standards (HTML 4, CSS, XML, JavaScript, and others).

Chapter 2. Getting Started

2.1. Hardware/software requirements

Hardware requirements:

1. CPU with Intel arch (Intel or AMD), min. 1 GHz
2. 256 Mb of RAM
3. Free hard disk space 100 Mb

Software requirements:

Windows version:

1. Microsoft Windows 2003/XP/Vista or higher with x86 arch;
2. Sun Microsystems Inc. Java JDK 1.5 or higher.

Linux version:

1. Linux Kernel 2.6;
2. GTK 2.10 with all dependent libraries;
3. Sun Microsystems Inc. Java JDK 1.5 32-bit version or higher.

If you install Firefox 3.0.1 package you will get all the necessary libraries. We recommend the Fedora Core 8 or higher but you can use any modern Linux distribution.

Mac OS X version:

1. Mac OS 10.5.2 Kernel;
2. Apple Java 1.5 32-bit version Installed.

2.2. JxBrowser Package Contents

The JxBrowser package contains the following files:

- JNIWrapper native libraries (jniwrap.dll on Windows platform, libjniwrap.so on Linux x86 platform, lib-jniwrap.jnilib on Mac OS X platform);
- JxBrowser library (jxbrowser.jar);
- JNIWrapper library (jniwrap.jar);
- WinPack library (winpack.jar) (Windows package only);
- MackPack library (mackpack.jar) (Mac OS X package only);
- Third-party Java libraries (MozillaInterfaces.jar, MozillaGlue.jar);
- Mozilla XULRunner application (xulrunner-windows, xulrunner-linux, xulrunner-mac);
- Mozilla XULRunner patches and build configs (xulrunner-build);

- JxBrowserDemo application and its source code;
- JxBrowser public API JavaDocs;
- JxBrowser License agreement (License.txt);
- Readme file (Readme.txt).

Chapter 3. Getting started

3.1. Creating and configuring Java project

We will demonstrate you creating and configuring a new project using Eclipce IDE, but this does not mean that you can't use other IDE. To configure your IDE refer to respective documentation.

At first create a new Java project. To use the JxBrowser you need the JxBrowser distribution package and license files - jniwrapper.lic and jxbrowser.lic. You can download the latest version of JxBrowser from www.teamdev.com web site. After you downloaded the JxBrowser package unpack it to any folder, and put the license files into `bin` subfolder.

Eclipse uses the `bin` folder to store its compiled files by default and JxBrowser uses this folder to store its native libraries like JNIWrapper. That is why we recommend renaming Eclipse directory for class files before using (for example into "eclipse_bin"). Otherwise, Eclipse will delete native libraries from the bin folder when you clear project build results.

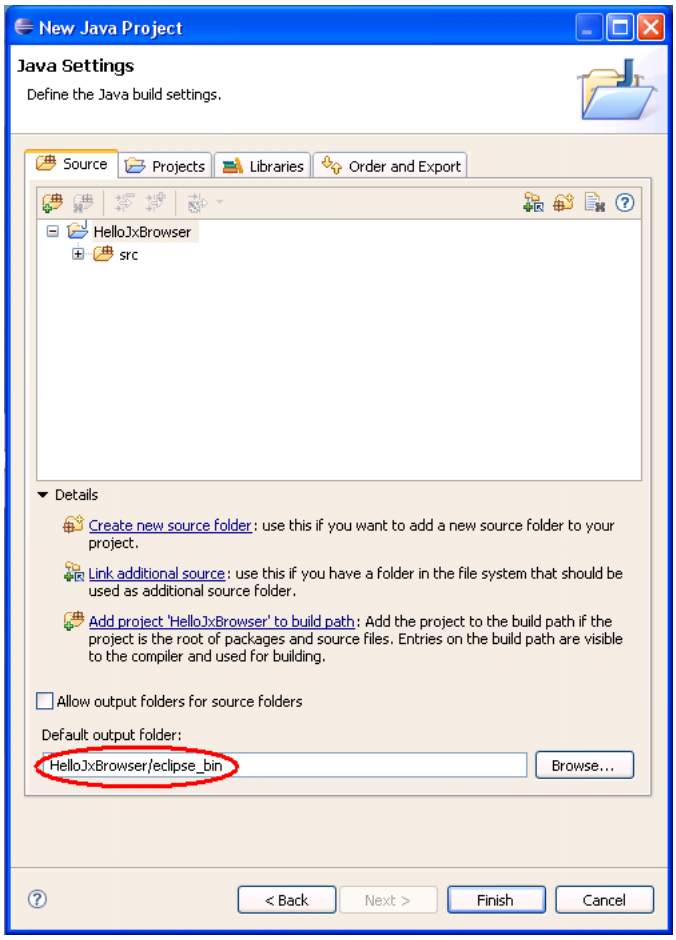

Figure 1. Configuring eclipse project

Copy the JxBrowser distribution folder into your new project folder. This is not obligatory but it makes the work process and writing of build scripts easier.

3.2. Configuring the Build Path

Please right click on the project in the project tree and configure your build path settings as shown in Figure 2:

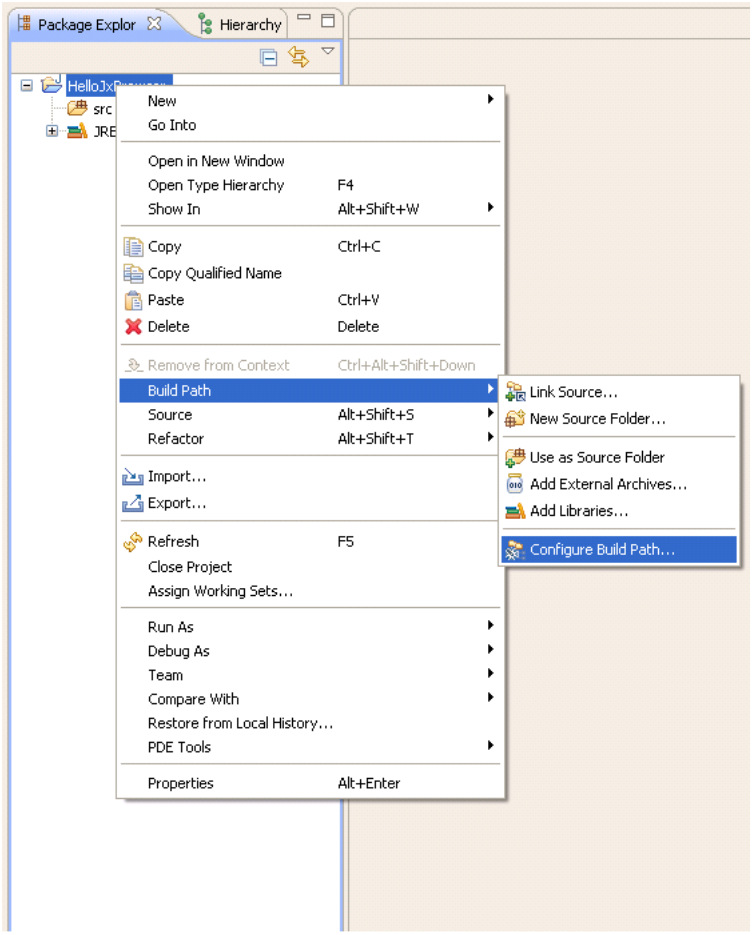

Figure 2. Access to build path configure dialog

Add the jxbrowser.jar, jniwrap-3.x.x.jar, Mozillainterfaces.jar, Mozillaglue.jar and winpack-3.x.jar or macpack-1.x.jar to build path. In case of Mac OS X you need to add the Apple Java extensions library. For this purpose you should create a new class folder and link it to the system folder with path /System Library/Java/. You can do it by pressing **Add Class Folder-> Create New Folder->Advanced** .

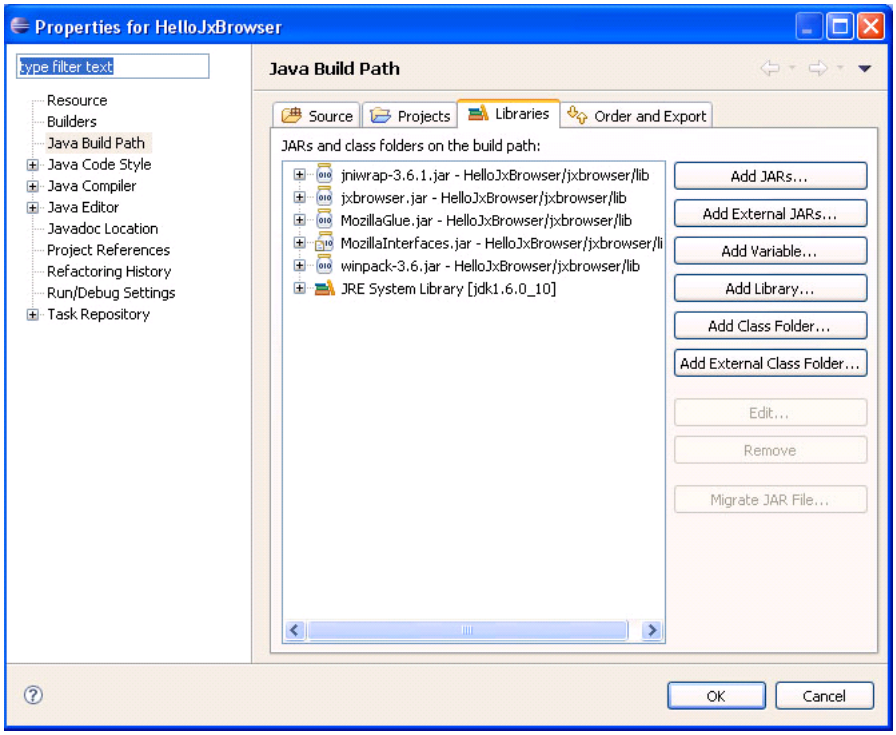

Figure 3. Java Build Path configuring

Chapter 4. Creating and Running “Hello World” Program for JxBrowser

In this chapter we will show - how to use the JxBrowser in your Java Swing applications. Here is a simple example program that creates a window with web browser component. To add the JxBrowser component in your Swing window, you should initialize the JxBrowser library. It means before using JxBrowser you have to start the Mozilla event loop and load the Mozilla native libraries in your JVM process. Then you should connect the Mozilla interfaces to AWT or Swing components. To initialize the Mozilla message loop thread you should call Xpcom.initialize(Xpcom.AWT) method. Then we recommend continuing working from Swing runner thread. In this case, the main thread, which serves main method, will end and will not consume system time and resources, but this is not mandatory. To invoke your code inside Swing runner thread you can use the SwingUtilites class. Now you need a web browser component. You can get the instance of this component by using WebBrowser.getComponent() method. This component is AWT heavyweight component, so be careful using it in your Swing programs. You can read about mixing AWT and Swing components more detailed at the following article. To create web browser instance you have to use the **WebBrowserFactory** class. Now you can download some HTML content inside the browser component. It can be done even if you do not have web server with testing page or some HTML document file. You will find more information about navigation and the content loading in the next chapter. Now we will use the WebBrowser.setContent method. Here is the whole code of “Hello World” program for JxBrowser:

```
import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import com.teamdev.jxbrowser.WebBrowser;
import com.teamdev.jxbrowser.WebBrowserFactory;
import com.teamdev.xpcom.Xpcom;

public class HelloWorld {

 public static void main(String[] args) {
  Xpcom.initialize(Xpcom.AWT);
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      WebBrowser browser = WebBrowserFactory.getInstance().createBrowser();
      browser.setContent("<html> <head></head>"+
                  "<body> <h1>Hello world</h1> </body> </html>",
                  "text/html");

      // Creating window for browser, and showing it
      JFrame browserFrame = new JFrame();
      browserFrame.setLayout(new BorderLayout());
      browserFrame.add(browser.getComponent());
      browserFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      browserFrame.setSize(640,480);
      browserFrame.setVisible(true);
    }
  });
 }

}
```

Ok, now run it. Open the Run Configuration dialog box and set JVM **java.library.path** to /bin folder and **GRE_HOME** to xulrunner folder (See Figures 4 and 5):

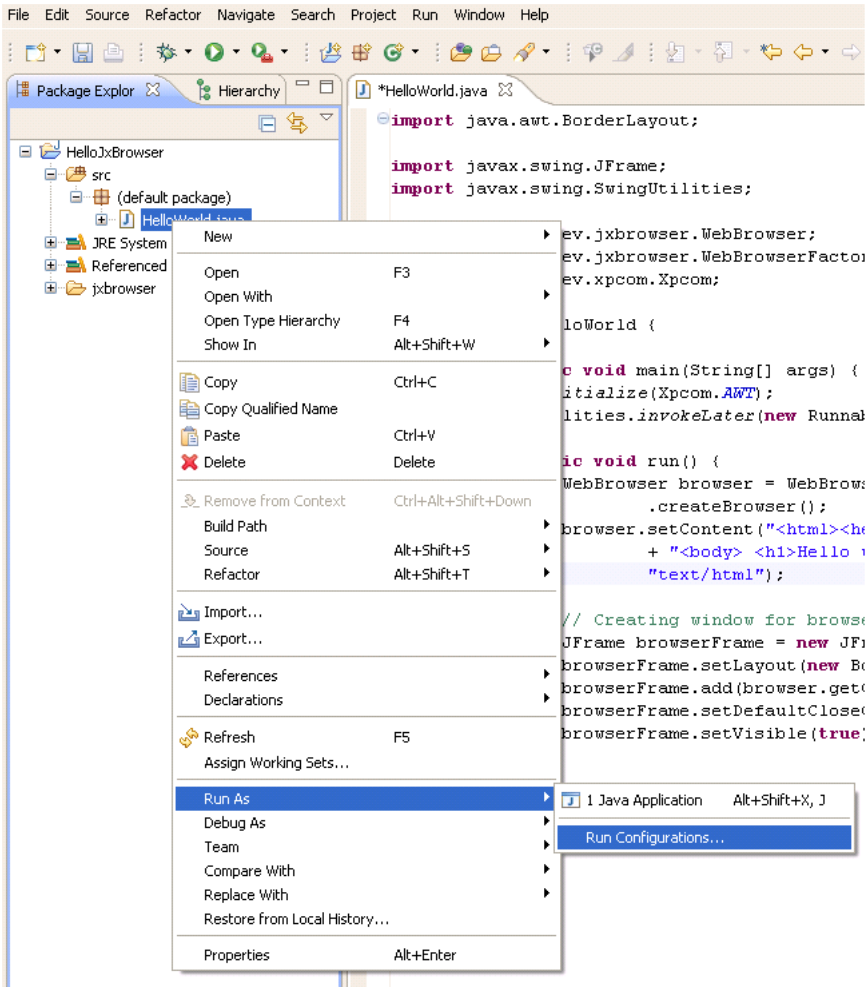

Figure 4. Access to Run Configurations Dialog box

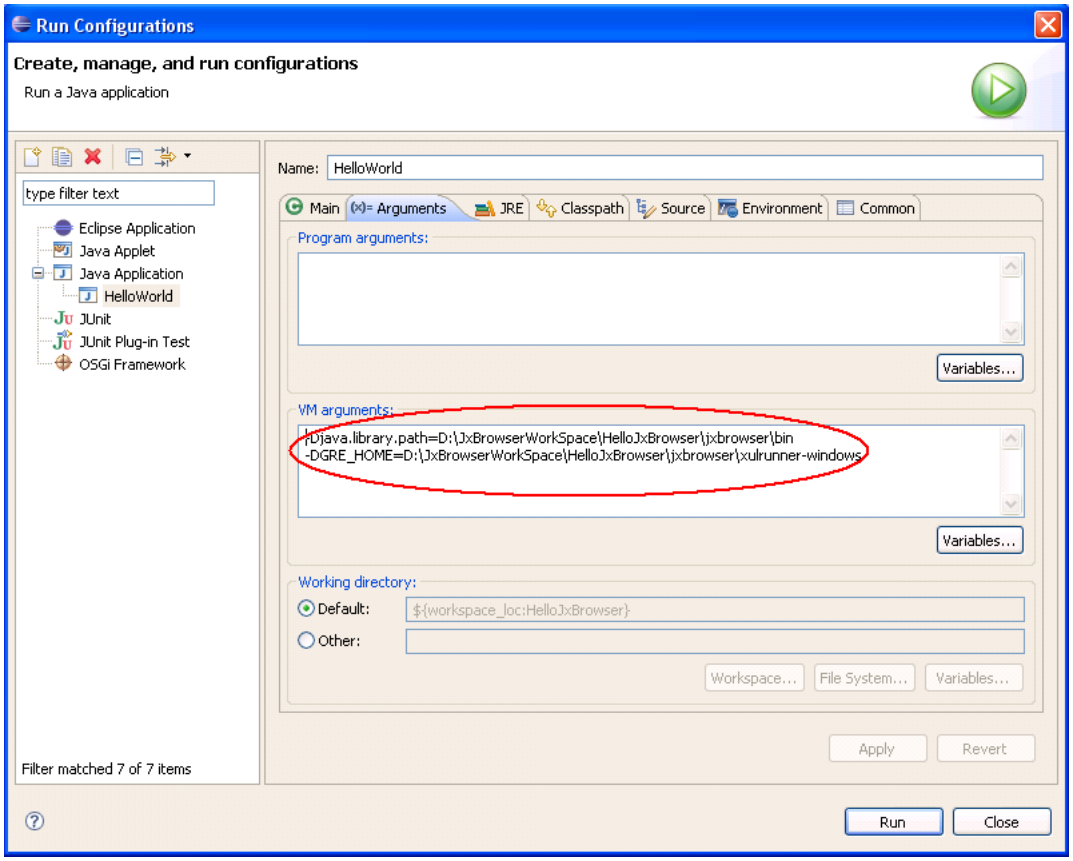

Figure5. Run configurations Dialog Box

Click on the Run button. If you have done everything correctly you will see “Hello World” message in browser window.

Chapter 5. Navigate Browser Functions

In the sample of “Hello world” program, we use the **setContent()** method of **WebBrowser** interface. However, in most cases, the functionality of this method is not enough, because it does not add the page into navigation history. So if you call

```
WebBrowser.setContent("<h1> Hello JxBrowser </ h1>", "text / html")
```

method you will not be able to return to “Hello World” page. Therefore, JxBrowser API has

```
WebBrowser.setContentWithContext (String content, String url, String contentType)
```

method.

If you use this method you can return to this page using the

```
WebBrowser.goBack()
```

method.

You can get the complete navigation history by calling

```
WebBrowser.getHistory()
```

method.

In order to load the page from the web server using network, you should use the WebBrowser.navigate(String url) method. You can use the **goBack()** and **goForward()** methods to navigate through history. You can also use the **stop()** method to stop downloading content. If you also want to add the post data information to some HTML forms on the page for the navigation you can use the other implementation of the

```
WebBrowser.navigate(String url, String postData)
```

method.

Chapter 6. Request Listeners

Sometimes you need to know what kind of operation your browser performs. For this purpose JxBrowser provides API to add special classes to WebBrowser instance. These classes must implement the **RequestListener** interface. The methods of these classes will be called when instance of the web browser changes its state. This API is very important and used quite often, so you should thoroughly review the documentation and the source code of JxBrowserDemo application. Web browser works in separate threads different from the main thread; you should know that all methods of RequestListener interface are invoked from Xpcom thread. Therefore, if you execute some code that uses Swing or AWT inside the **RequestListener** methods, it must be done within Swing runner thread. In most cases, the **SwingUtiltes** class static methods such as **invokeAndWait(Runnable run)** and **invokeLater(Runnable run)** perfectly suit for executing any code inside Swing worker thread. You can also use similar methods of **Xpcom** class for working in Xpcom thread. Here is the example of using API for **RequestListener** interfaces:

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.SwingUtilities;

import com.teamdev.jxbrowser.WebBrowser;
import com.teamdev.jxbrowser.WebBrowserFactory;
import com.teamdev.jxbrowser.event.LocationEvent;
import com.teamdev.jxbrowser.event.ProgressEvent;
import com.teamdev.jxbrowser.event.RequestListener;
import com.teamdev.jxbrowser.event.SecurityEvent;
import com.teamdev.jxbrowser.event.StateEvent;
import com.teamdev.jxbrowser.event.StatusEvent;
import com.teamdev.xpcom.Xpcom;

/**
   A frame that contains a WebBrowser component to rendering http://www.teamdev.com,
   a progress bar, a label for status and a label for current location of WebBrowser.
*/
public class RequestListenerDemo extends JFrame {

   private static final String THE_URL = "http://www.teamdev.com";

   final JProgressBar progressBar = new JProgressBar(0);

   final JLabel statusLabel = new JLabel();

   final JLabel locationLabel = new JLabel();

   /**
      Listener for receiving various events associated with the loading of asynchronous requests.
   */
   RequestListener requestListener = new RequestListener() {

      /**
         Invoked when the location of the WebBrowser is changed.
      */
      public void locationChanged(final LocationEvent e) {
         SwingUtilities.invokeLater(new Runnable() {
            public void run() {
               locationLabel.setText("Location :" + e.getLocation());
            }
```

```
        });
    }

    /**
       Invoked when the progress is changed for one of the requests.
    */
    public void progressChanged(final ProgressEvent e) {
       SwingUtilities.invokeLater(new Runnable() {
          public void run() {
             progressBar.setMaximum((int) e.getMaxTotalProgress());
             progressBar.setValue((int) e.getCurTotalProgress());
          }
       });
    }

    /**
       Invoked on security transitions (eg HTTP -> HTTPS, HTTPS -> HTTP, FOO -> HTTPS)
       and after document load completion.
    */
    public void securityChanged(SecurityEvent e) {
    }

    /**
       Notification indicating the state is changed for one of the requests.
    */
    public void stateChanged(StateEvent e) {
       SwingUtilities.invokeLater(new Runnable() {
          public void run() {
             statusLabel.setText("Done");
          }
        });
    }

    /**
       Changing statusLabel when the status of a request is changed.
    */
    public void statusChanged(final StatusEvent e) {
       SwingUtilities.invokeLater(new Runnable() {
          public void run() {
             statusLabel.setText(e.getMessage());
          }
       });
    }
};

/**
   Creating UI and adding RequestListener for WebBrowser
*/
void myMain() throws Throwable {

    /**
       Create a browser instance.
    */
    WebBrowser browser = WebBrowserFactory.getInstance().createBrowser();

    setLayout(new BorderLayout());
    JPanel statuPanel = new JPanel();
    statuPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
    statuPanel.add(progressBar);
    statuPanel.add(statusLabel);
    statuPanel.add(locationLabel);

    add(statuPanel, BorderLayout.SOUTH);
    add(browser.getComponent(), BorderLayout.CENTER);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(640, 480);
        setVisible(true);

        /**
           Add RequestListener for WebBrowser
        */
        browser.addRequestListener(requestListener);

        /**
           Navigate to a resource specified by a URL
           or to the file identified by a full path.
        */
        browser.navigate(THE_URL);
    }

    public static void main(String[] args) {

        /**
           Initialize XPCOM system for AWT/Swing UI.
        */
        Xpcom.initialize(Xpcom.AWT);

        /**
           WebBrowser works in separate threads different from the main thread
        */
        SwingUtilities.invokeLater(new Runnable() {
           public void run() {
              RequestListenerDemo demo = new RequestListenerDemo();
              try {
                 demo.myMain();
              } catch (Throwable t) {
                 t.printStackTrace();
              }
           }
        });

    }
}
```

The **WebBrowser** instance can have as many **RequestListener** interface instances as you want if you have enough free memory. If you need to use only one or two methods of RequestListener interface then you can use the **RequestAdapter** abstract class that implements the **RequestListener** interface and overrides necessary methods.

Chapter 7. Multithreading model

All programs using JxBrowser library utilize at least two threads. These threads accomplish message dispatching, so please use JxBrowser library carefully and follow the recommendation below: The Xpcom thread performs all operations with Mozilla interfaces and dispatches the windows subsystem messages. The **RequestListener** methods are invoked from native code inside Xpcom thread. Therefore, you should perform all operations that use Swing or AWT and that will be executed inside the methods of classes, which implement the **RequestListener** interface, in Swing runner thread. If you do not accomplish this condition, you can have an error difficult for detecting. Swing runner thread performs all operations with the GUI. Usually the code that you write using Swing is performed inside this thread. SwingUtitilites class is necessary for working through this thread. Moreover, here are some recommendations to work with threads.

If you need to perform some operations using the Mozilla API, it is better to do it within Xpcom thread as described below:

1. All operations with GUI are invoked from Swing runner thread;
2. Do not stop either Swing runner or Xpcom threads if you do not need to synchronize their work;
3. There is no sense in invoking operations that take a long time, for example: parsing of document DOM tree or sorting array with 100 millions elements in Xpcom or in the Swing runner threads. It can result in delays with processing messages from windowing system. Do it in some other thread;
4. Do not abuse invokeAndWait methods of Xpcom and SwingUtilites classes. These methods stop the thread that invokes them while Xpcom or Swing runner does not finish the operation from Runnable parameter of these functions. Do not use these methods if you do not have real necessity to stop current thread until this method is finished.

Chapter 8. Browser modes

The web browser has several modes:

1. HTML View mode;
2. View HTML as Source mode;
3. Print Preview mode.

The default mode is HTML View, see the following sample to understand how to change the browser mode. After switching to the View HTML as Source mode, you cannot return to HTML View mode. To move up to Print Preview mode you should wait until the browser loads a page, otherwise deadlock may happen but usually it leads to browser scroll bars becoming inaccessible.

Mode sample:

```
import java.awt.BorderLayout;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import com.teamdev.jxbrowser.WebBrowser;
import com.teamdev.jxbrowser.WebBrowserFactory;
import com.teamdev.jxbrowser.event.RequestAdapter;
import com.teamdev.jxbrowser.event.RequestListener;
import com.teamdev.jxbrowser.event.StateEvent;
import com.teamdev.jxbrowser.printing.WebBrowserPrinting;
import com.teamdev.xpcom.Xpcom;

public class SourceModesSample extends JFrame {

 private static final String THE_URL = "http://www.teamdev.com/";

 final AtomicBoolean loaded = new AtomicBoolean(false);

 RequestListener requestListener = new RequestAdapter() {

  @Override
  public void stateChanged(StateEvent e) {
   if (!e.isLoadingDocument() && e.isNetwork()) {
    synchronized (loaded) {
     loaded.set(true);
     loaded.notifyAll();
    }
   }
  }

 };

 void myMain() throws Throwable {
  WebBrowser browser = WebBrowserFactory.getInstance().createBrowser();

  setLayout(new BorderLayout());
  add(browser.getComponent(), BorderLayout.CENTER);

  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(640, 480);

  browser.addRequestListener(requestListener);

  browser.navigate(THE_URL);
```

```
  // wait while browser loading the content
  synchronized (loaded) {
   if(!loaded.get()) {
    loaded.wait();
   }
  }
  browser.displayAsSource();

  setVisible(true);
  Thread.sleep(1000);
  setVisible(false);

  // now go to print preview mode
  WebBrowserPrinting printAPI = browser.getPrinting();
  printAPI.setPrintPreviewMode(true);

  setVisible(true);
  Thread.sleep(1000);
  setVisible(false);

  // now return to view as source mode
  printAPI.setPrintPreviewMode(false);
  setVisible(true);
}

public static void main(String[] args) {
  Xpcom.initialize(Xpcom.AWT);
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      SourceModesSample demo = new SourseModesSample();
      try {
       demo.myMain();
      } catch (Throwable e) {
        e.printStackTrace();
      }
    }
  });
}

}
```

Chapter 9. Printing

The JxBrowser also contains a powerful printing API using which you can access most of printer options. This guide does not describe it in detail, but you can find it in the JavaDoc documentation as well as in the source code of our JxBrowserDemo application. In the following example, we only explain the principle of working with printing API. To start the following sample program you need to have at least one printer installed in your system. The Linux version also supports printing to PDF file and to Post Script file. In order to print the document you don't need to switch to the Print Preview mode before you print the document. If you need special dialog boxes for print and print preview, you can get them from JxBrowserDemo application.

The Printing sample:

```
import com.teamdev.xpcom.Xpcom;
import com.teamdev.jxbrowser.WebBrowser;
import com.teamdev.jxbrowser.WebBrowserFactory;
import com.teamdev.jxbrowser.printing.*;
import com.teamdev.jxbrowser.event.*;

import javax.swing.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.awt.*;

public class PrintExample {
   /**
    * Navigates {@code browser} to a specified {@code url} and waits
    * until the document is loaded completely.
    */
   private static void navigateAndWait(WebBrowser browser, String url) {
      final AtomicBoolean navigateInProgress = new AtomicBoolean(true);
      RequestListener requestListener = new RequestAdapter() {
         private String rootRequest;

         @Override
         public void locationChanged(LocationEvent event) {
            navigateInProgress.getAndSet(true);
            if (!event.getLocation().equals("about:blank")) {
               // if the page is not 'about:blank' then init rootRequest
               rootRequest = event.getLocation();
            }
         }

         @Override
         public void stateChanged(StateEvent event) {
            boolean isRootRequestState = event.getRequestUrlName().equals(rootRequest);
            if (event.isLoadingCompleted() && isRootRequestState) {
               navigateInProgress.getAndSet(false);
               Xpcom.unlock();
            }
         }
      };
      browser.addRequestListener(requestListener);
      browser.navigate(url);
      // wait until the document is loaded completely
      while (navigateInProgress.get()) {
         Xpcom.lock();
      }
      browser.removeRequestListener(requestListener);
   }

   public static void main(String[] args) {
      Xpcom.initialize();
      WebBrowser browser = WebBrowserFactory.getInstance().createBrowser();
```

```
        JFrame frame = new JFrame("Mozilla need a window");
        frame.setLayout(new BorderLayout());
        frame.add(browser.getComponent(), BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);

        navigateAndWait(browser, "http://www.google.com");

        // Get the printing API for the web browser instance
        WebBrowserPrinting printing = browser.getPrinting();

        // Print with default printer and settings
        // printing.print(null);

        // You can set settings of printer yourself
        PrintSettings printSettings = new PrintSettings();
        printSettings.setPrinterName(printing.getDefaultPrinterName());
        printSettings.setHeaders(new HeaderFooter("header_left", "header_center", "header_right"));
        printSettings.setFooters(new HeaderFooter("footer_left", "footer_center", "footer_right"));
        FrameSettings frameSettings = new FrameSettings(
                FrameSettings.HowToEnableFrameUI.EnableAsIsAndEach,
                FrameSettings.PrintFrameTypeUsage.AsIs);
        printSettings.setFrameSettings(frameSettings);
        printSettings.setNumCopies(1);
        printSettings.setOrientation(PrintSettings.PageOrientation.Portrait);
        printSettings.setPaperSize(PaperSize.A4);
        printSettings.setPrintBackgroundColors(true);
        printSettings.setPrintBackgroundImages(true);
        printSettings.setPrintInColor(true);
        printSettings.setRange(new PageRange(PageRange.RangeType.All));
        printSettings.setScaling(1.0);
        printSettings.setShrinkToFit(false);

        //Printing with defined settings.
        printing.initPrintSettings(printSettings);
        printing.print(new PrintProgressAdapter() {
          @Override public void printingFinished() {
            System.out.println("Print is finished");
          }
        });
    }
}
```

Chapter 10. Mixing JxBrowser API and Mozilla interfaces

Sometimes it is necessary to use Mozilla interfaces in your program. It can be useful if you want to add some additional functionality that is not initially provided by JxBrowser API or if you need a native component such as a system Open File dialog box, etc. You should follow the main rule: “Perform all operations with Mozilla interfaces inside Xpcom thread only”. Next source code samples demonstrate how to use the Mozilla interfaces in combination with JxBrowser API.

1. Getting Mozilla service:

```
final String ourURI = "http://www.teamdev.com";
 Xpcom.invokeLater(new Runnable() {
  public void run() {
   nsIServiceManager sm = Mozilla.getInstance().getServiceManager();
   nsIIOService io = (nsIIOService) sm.getServiceByContractID(
                                "@mozilla.org/network/io-service;1",
                                nsIIOService.NS_IIOSERVICE_IID);
   nsIURI uri = io.newURI(ourURI, null, null);
  }
 });
```

2. Creating Mozilla component:

```
Xpcom.invokeLater(new Runnable() {
 public void run() {
  nsIComponentManager cm = Mozilla.getInstance().getComponentManager();
  nsIFilePicker filePicker = (nsIFilePicker) cm.createInstanceByContractID(
                                "@mozilla.org/filepicker;1", null,
                                nsIFilePicker.NS_IFILEPICKER_IID);
 }
});
```

3. Accessing to **nsIWebBrowser** service from **WebBrowser** interface and getting **nsIDOMWindow** and **nsIDOMDocument** Mozilla interfaces:

```
Xpcom.invokeLater(new Runnable() {
 public void run() {
  // the browser is instance of your WebBrowser
  nsIWebBrowser mozBrowser = ((MozillaWebBrowser) browser).getWebBrowser();
  nsIDOMWindow window = mozBrowser.getContentDOMWindow();
  nsIDOMDocument document = window.getDocument();
  // Now you can get reference to any DOM element from document
 }
});
```

Chapter 11. JavaScript

You can perform a JavaScript code directly from your Java code. For this purpose, there is the special feature evaluateScript(String script) method. Here is an example of how to fill in the Google search field using WebBrowser.evaluateScript() method.

```
import java.awt.BorderLayout;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import com.teamdev.jxbrowser.WebBrowser;
import com.teamdev.jxbrowser.WebBrowserFactory;
import com.teamdev.jxbrowser.event.RequestAdapter;
import com.teamdev.jxbrowser.event.RequestListener;
import com.teamdev.jxbrowser.event.StateEvent;
import com.teamdev.xpcom.Xpcom;


public class EvaluateScriptDemo extends JFrame {

 private static final String THE_URL = "http://www.google.com/";

 final AtomicBoolean loaded = new AtomicBoolean(false);

 RequestListener requestListener = new RequestAdapter() {
  @Override
  public void stateChanged(StateEvent e) {
   if (!e.isLoadingDocument() && e.isNetwork()) {
    synchronized (loaded) {
     loaded.set(true);
     loaded.notifyAll();
    }
   }
  }
 };

 void myMain() throws Throwable {
  final WebBrowser browser = WebBrowserFactory.getInstance().createBrowser();

  setLayout(new BorderLayout());
  add(browser.getComponent(), BorderLayout.CENTER);

  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(640, 480);

  browser.addRequestListener(requestListener);

  browser.navigate(THE_URL);

  // wait while browser loading the content
  synchronized (loaded) {
   if (!loaded.get()) {
    loaded.wait();
   }
  }

  browser.evaluateScript("document.f.q.value = 'TeamDev' ");

  setVisible(true);
 }
```

```
 public static void main(String[] args) {
  Xpcom.initialize(Xpcom.AWT);
  SwingUtilities.invokeLater(new Runnable() {
   public void run() {
     EvaluateScriptDemo demo = new EvaluateScriptDemo();
     try {
      demo.myMain();
     } catch (Throwable e) {
      e.printStackTrace();
     }
   }
  });
 }

}
```

You can also use console service API to check your JavaScript, CSS, HTML etc. code for errors. There is the simple example of using the console service below.

```
private ConsoleListener listener = new ConsoleListener() {
  public void actionPerformed(MessageEvent event) {
    if (event instanceof ScriptErrorEvent) {
        // your actions to JavaScript's errors
    }
     System.err.println(event.getMessage());
  }
};

{
  ConsoleService service = ConsoleService.getConsoleService();
  service.registerConsoleListener(listener);
}
```

Chapter 12. Mozilla plug-ins in JxBrowser

You can use some Mozilla Firefox plug-ins in JxBrowser. We cannot guarantee the correct work of plug-ins made by unknown vendors, though we can confirm that you can safely use plug-ins produced by Adobe (Adobe Flash and Adobe Acrobat Reader (acroreader) and Java plug-ins.

For using Mozilla Firefox plug-ins in JxBrowser you need to have Mozilla Firefox 3 installed.

For Microsoft Windows:

Please setup FireFox 3, after setup Adobe Reader

Note: Adobe Acrobat Reader (8 or higher version). You can download it for free at http://www.adobe.com/

For Linux:

Setup Adobe Acrobat Reader (if Firefox is installed).

Note: We advise to use installation packages for your system (We tested them on Fedora Core 7/8, Ubuntu and Kubuntu).

Then copy acroreader plug-in lib nppdf.so from `/usr/lib/mozilla/plugins` to your `xulrunner-linux/plugins` directory.

For Mac OS X

Same as for Linux.

Chapter 13. Using JxBrowser with Java Web Start

This section describes the way of deploying your application that uses JxBrowser with the help of Java Web Start (JWS).

To deploy your application with the help of JWS, please follow the next simple steps:

1. Pack application classes to JAR file(s);
2. Put the runtime license files (jniwrap.lic and jxbrowser.lic) to the META-INF folder of an application JAR file;
3. Sign the application JAR files;
4. Prepare the JNLP start-up file;
5. Upload the JAR files to your Web-server.

Consider using of the JxBrowser in JWS taking the JxBrowserDemo application as an example.

The start-up file (JxBrowser.jnlp) can look like that:

```
<jnlp spec="1.5+" codebase="http://www.teamdev.com/downloads/demo" href="JxBrowser.jnlp">
  ...
  <security>
    <all-permissions/>
  </security>
  ...
  <resources>
    <j2se version="1.5+" initial-heap-size="64m"/>

    <jar href="lib/licenses.jar"/>
    <jar href="lib/bin.jar"/>
    <jar href="lib/demo.jar" main="true"/>
    <jar href="lib/log4j-1.2.8.jar"/>
    <jar href="lib/swing-layout-1.0.jar"/>
    <jar href="lib/AbsoluteLayout.jar"/>
    <jar href="lib/aclibico-2.1.jar"/>
    <jar href="lib/jniwrap-3.7.1.jar"/>
    <jar href="lib/MozillaInterfaces.jar"/>
    <jar href="lib/MozillaGlue.jar"/>
    <jar href="lib/jxbrowser.jar"/>

    <property name="javaws.jxbrowser.com.teamdev.www" value="true"/>

  </resources>
  <resources os="Windows">
    <jar href="lib/winpack-3.6.jar"/>
    <jar href="xulrunner/xulrunner-win32.jar"/>
  </resources>
  <resources os="Linux">
    <jar href="xulrunner/xulrunner-linux-i686.jar"/>
  </resources>
  <resources os="Mac OS X">
    <jar href="lib/macpack-1.0.jar"/>
    <jar href="xulrunner/xulrunner-mac-un-i386-ppc.jar"/>
  </resources>
  <application-desc main-class="com.teamdev.xpcom.mozilla.demo.JxBrowserDemo"/>
</jnlp>
```

Important: Pay your special attention. Start-up file must contain the following options:

```
...
<security>
  <all-permissions/>
</security>
...
<property name="javaws.jxbrowser.com.teamdev.www" value="true"/>
...
```

These options are designed for getting access to client side for unpacking of the xulrunner application. Using the **javaws.jxbrowser.com.teamdev.www** property JxBrowser defines an application as one that uses the JWS technology. Then JxBrowser unpacks the **xulrunner** application into the **deployment.user.tmp** directory on the client side. Thus the **GRE_HOME= ${deployment.user.tmp}** property will be set up on the client side that starts this application using the JWS technology. Following this path, the **xulrunner** application will be unpacked on the client side.

The following files are listed in the start-up file (JxBrowser.jnlp):

- licenses.jar - archive includes the license files (jniwrap.lic, jxbrowser.lic);
- bin.jar - archive includes the files (jniwrap.dll, libjniwrap.jnilib, libjniwrap.so);
- demo.jar, log4j-1.2.8.jar, swing-layout-1.0.jar, AbsoluteLayout.jar, aclibico-2.1.jar - the demo application (JxBrowserDemo
- jxbrowser.jar - archive of the JxBrowser library.
- xulrunner-win32.jar, xulrunner-linux-i686.jar, xulrunner-mac-un-i386-ppc.jar - the Mozilla XULRunner application for Microsoft Windows, Linux, Mac OS X. These jars should have the following strict structures.

  xulrunner-win32.jar

  xulrunner-win32.jar/META-INF

  xulrunner-win32.jar/xulrunner-win32.zip

  xulrunner-win32.jar/xulrunner-win32.zip/xulrunner

  xulrunner-win32.jar/xulrunner-win32.zip/xulrunner/chrome

  xulrunner-win32.jar/xulrunner-win32.zip/xulrunner/...

  xulrunner-win32.jar/xulrunner-win32.zip/xulrunner/xulrunner.exe

  xulrunner-linux-i686.jar

  xulrunner-linux-i686.jar/META-INF

  xulrunner-linux-i686.jar/xulrunner-linux-i686.zip

  xulrunner-linux-i686.jar/xulrunner-linux-i686.zip/xulrunner

  xulrunner-linux-i686.jar/xulrunner-linux-i686.zip/xulrunner/chrome

  xulrunner-linux-i686.jar/xulrunner-linux-i686.zip/xulrunner/...

  xulrunner-linux-i686.jar/xulrunner-linux-i686.zip/xulrunner/xulrunner-stub

  xulrunner-mac-un-i386-ppc.jar

  xulrunner-mac-un-i386-ppc.jar/META-INF

  xulrunner-mac-un-i386-ppc.jar/xulrunner-mac-un-i386-ppc.zip

  xulrunner-mac-un-i386-ppc.jar/xulrunner-mac-un-i386-ppc.zip/xulrunner

  xulrunner-mac-un-i386-ppc.jar/xulrunner-mac-un-i386-ppc.zip/xulrunner/chrome

  xulrunner-mac-un-i386-ppc.jar/xulrunner-mac-un-i386-ppc.zip/xulrunner/...

xulrunner-mac-un-i386-ppc.jar/xulrunner-mac-un-i386-ppc.zip/xulrunner/xulrunner-bin

All JAR files should contain similar signatures. For this purpose, you can use the following ant templates:

```
<project name="Sample" default="build" basedir=".">

  <target name="binjar">
    <property name="binjarName" value="bin.jar"/>
    <jar destfile="${binjarName}">
     <fileset dir="${srcPath}/bin" includes="*.dll"/>
     <fileset dir="${srcPath}/bin" includes="*.jnilib"/>
     <fileset dir="${srcPath}/bin" includes="*.so"/>
    </jar>
    <signjar jar="${binjarName}" alias="your_alias" keystore="your_keystore"
    keypass="your_keypass" storepass="your_storepass"/>
  </target>

  <target name="licjar">
    <property name="licensesjarName" value="licenses.jar"/>
    <jar destfile="${licensesjarName}">
      <metainf dir="${licensesPath}" includes="*.lic"/>
    </jar>
    <signjar jar="${licensesjarName}" alias="your_alias" keystore=" your_keystore"
    keypass="your_keypass" storepass="your_storepass"/>
  </target>

</project>
```

where:

${srcPath}/bin - path to the following files:jniwrap.dll, libjniwrap.jnilib, libjniwrap.so;

your_alias, your_keystore, your_keypass, your_storepass - signjar parameters. You can see their description in Apache Ant User Manual at http://ant.apache.org

Using JWS technology you can start the demo application (JxBrowserDemo) at: Online Demo

Chapter 14. Support

14.1. Reporting Problems

If you have any questions regarding JxBrowser, please e-mail us to: jxbrowser-support team

14.2. JxBrowser Forum

If you want to discuss any topics related to JxBrowser, please visit our support forum at:

JxBrowser forum

14.3. Troubleshooting

If you have any problems with using JxBrowser, you can submit your request using our online Support Request Form at:

Trouble Report From

Chapter 15. Where to Get a New Version

You can get the latest version of JxBrowser at: TeamDev Ltd. web site