

1

Riatrix4Applet Standalone Application

Riatrix4Applet applications normally run inside the browser. If you are doing a smooth migration from conventional HTML legacy applications, they have to run inside the browser because only then can Riatrix4Applet be integrated with the existing legacy HTML applications.¹

From a development point of view, creating a standalone Riatrix4Applet application is not much different from creating a browser based Riatrix4Applet application. You just use RiatrixApplication instead of RiatrixApplet and you have create your own frame window. The first example is a console application. The second example is a desktop application and an applet at the same time.

1.1 A Standalone Console Application

1.2 Service

The single service provided is a single method that returns the server's system properties.

1.2.1 Frontend

The main static method creates a new application object and invokes its constructor with the command line arguments. The constructor does all the work: it invokes the server and iterates through the properties, which it then prints to the console.

```
public class Riatrix4AppletStandaloneConsole
    extends RiatrixApplication {
    private static final long serialVersionUID = 1L;
```

¹Note that there is no sample for interaction between HTML and Riatrix4Applet applications as there are no special requirements for Riatrix4Applet applications (just use the JSObject interfaces as usual). The needs and complexity for such integration can range from mere parameter passing to tight coupling like Riatrix4Applet forms inside HTML forms.

```
@Services(allocation=Allocation.Global,
         implementation=SystemPropertyServicesImpl.class)
static SystemPropertyServices sps;

public Riatrax4AppletStandaloneConsole() {
    super();
}

/** Directly prints the results to the console */
public Riatrax4AppletStandaloneConsole(String[] args) {
    Properties serverProperties = getSystemProperties();
    for (Entry<Object, Object> e : serverProperties.entrySet()) {
        System.out.println(e.getKey() + ": " + e.getValue());
    }
}

public Properties getSystemProperties () {
    // this is a remote call:
    return sps.getSystemProperties();
}

public static void main(String[] args) {
    new Riatrax4AppletStandaloneConsole(args);
}

protected URL getRealBackendCodeBase() {
    try {
        return new URL(
            "http://localhost:8080/StandaloneApplication/service/");
    } catch (MalformedURLException e) {
        throw new IllegalStateException(e); // how???
    }
}
}
```

In the example, the application class is a subclass of `RiatraxApplication` and therefore, the necessary `RiatraxApplication` object is created as part of the creation of the application object in `main()`.

1.2.2 Providing the Backend Code Base

When running inside a browser or Sun's `AppletViewer`, the backend code base is specified in a the parameter definition. A standalone application cannot be expected to have such a source for parameter definitions. In a standalone application, you override `RiatraxApplication`'s `getRealBackendCodeBase` method where you return the URL, where the backend is available. It is the same URL that is provided for browser based apps. You can even access the same backend from browser and from a standalone application.

```
protected URL getRealBackendCodeBase() {
```



```

try {
    return new URL(
        "http://localhost:8080/StandaloneApplication/service/");
} catch (MalformedURLException e) {
    throw new IllegalStateException(e); // how???
}
}

```

The application can be started either on the console or in your IDE running the `Riatrix4AppletStandaloneConsole`'s main method. Before you do so, the application's WAR file needs to be deployed on the local server (since the application attempts to connect to localhost:8080).

1.2.3 Result

When running the application, the server has to be started first and then the application can be started as a normal Java SE application on the command line. It will print the server's properties, including those defined by the application server:

```

...
sun.cpu.isalist: pentium_pro+mmx pentium_pro
pentium+mmx pentium i486 i386 i86
package.definition: sun., java., org.apache.catalina.,
org.apache.coyote., org.apache.tomcat., org.apache.jasper.
sun.os.patch.level: Service Pack 2
java.version: 1.6.0_05
sun.management.compiler: HotSpot Client Compiler
...

```

1.3 A Desktop Standalone Application

The console application example illustrates the wiring that has to be set up to access the back-end. The desktop application displays a `JTable` with the same information. To not have to rewrite everything, the code from `RIATraxSampleStandalone` is simply reused:

```

@MainPanel(name="index")
public class Riatrix4AppletStandaloneDesktop extends RiatrixApplet {
    private static final long serialVersionUID = 1L;

    public void initContent() {
        Riatrix4AppletStandaloneConsole standalone =
            new Riatrix4AppletStandaloneConsole();
        Properties serverProperties = standalone.getSystemProperties();
        Object[][] rowdata = new Object[serverProperties.size()][2];

        int i = 0;
        for (Entry<Object, Object> e : serverProperties.entrySet()) {
            rowdata[i][0] = e.getKey();
            rowdata[i][1] = e.getValue();
        }
    }
}

```



```

        i++;
    }

    JTable table = new JTable(rowdata, new Object[]{"Key", "Value"});
    JScrollPane scrollPane = new JScrollPane(table);
    add(scrollPane, BorderLayout.CENTER);
}
}

```

Key	Value
sun.boot.library.path	C:\Program Files\Java\jre1.6.0_01\bin
browser.version	1.06
java.vendor.url	http://java.sun.com/
java.runtime.version	1.6.0_01-b06
java.vm.version	1.6.0_01-b06
java.home	C:\Program Files\Java\jre1.6.0_01
java.vm.name	Java HotSpot(TM) Client VM
package.restrict.access.sun	true
package.restrict.definition.sun	true
user.dir	C:\devel\workspace\StandaloneConsole\build\classes
java.version	1.6.0_01
java.awt.graphicsenv	sun.awt.Win32GraphicsEnvironment
java.vm.specification.version	1.0
sun.desktop	windows
os.arch	x86
java.security.policy	java.policy.applet
sun.cpu.endian	little

Applet gestartet

The main point of this example is to show that Riatrax4Applet can be combined with arbitrary other application frameworks – not only on the server, but also on the client side. This is not limited to Swing based frameworks either. It is just as easy to write a Riatrax4Applet Eclipse RCP as a JDesktop application. Deployment mechanisms can vary and Riatrax4Applet provides Web Start support.

Moreover, the application can also be accessed in the browser as an index.html file exists that references the applet. That is what makes Riatrax4Applet so flexible: it can be used as a console application, locally on the desktop or in the browser as an applet with no additional overhead.

1.3.1 Running the applications in Eclipse

Download the project from <http://www.fybit.com/download/riatrax4applet/source/StandaloneApplication.zip> and import it to the Eclipse IDE. Drag the build.xml to the Ant view and doubleclick the *rebuild* target. Right click on the project → Run As → Run on Server. If your browser does not open automatically, go to <http://localhost:8080/StandaloneApplication/>.

To run the console application, right click on Riatrax4AppletStandaloneConsole → Run As → Java Application. Eclipse's console view will output the data retrieved from the server.



To run the standalone Java application on the desktop, right click on Riatrax4AppletStandaloneDesktop → Run As → Java Applet. You then see the application open in its own window.



