

FYBIT Riatrix Project Setup for Eclipse

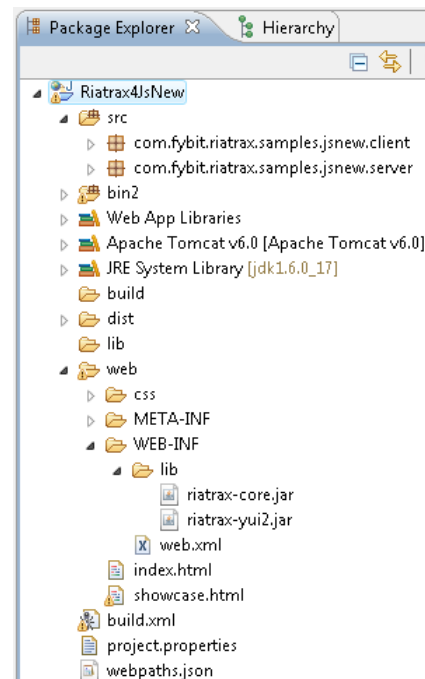
This guide explains you how to work with Fybit Riatrix to develop powerful web applications. The guide uses Riatrix4JsNew.zip for the explanation and assumes that you have completed the setup guide.

Project Layout, Compile, Launch

Project Layout

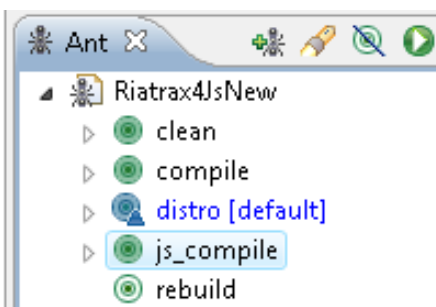
The Riatrix4JsNew project uses a standard Java project layout. The src folder contains the Java sources. All files in the web directory are available in the web application, e.g. index.html. The web/WEB-INF/lib directory contains the Fybit JAR libraries.

The directory bin2 is created by the Fybit engine and contains classes for remoting as well as the generated JavaScript code. Normally, you don't have to look at or edit these files.



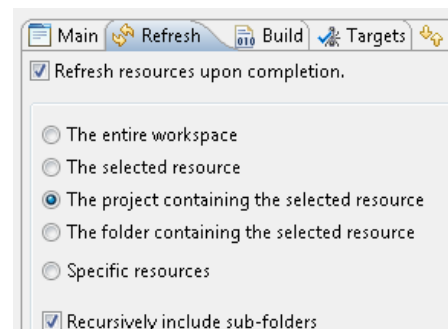
Building the application

To compile the application, you need to open the Ant view: Window → Show View → Other → Ant → Ant. Then, drag the build.xml file to the Ant view.



- clean: removes all generated files from bin2
- compile: creates the classes needed for remoting
- distro: creates a WAR file to be deployed
- js_compile: compiles the client parts to JavaScript
- rebuild: does clean and js_compile in one step

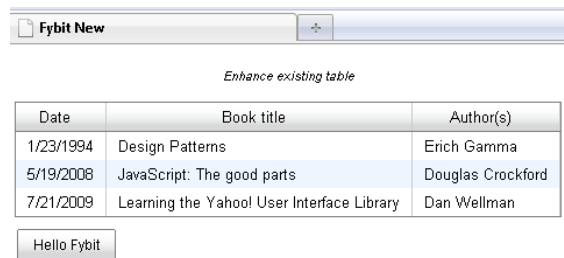
The Ant target you use during development is js_compile. We recommend to automatically refresh the project upon building with Ant, which can be configured as follows: Right click on js_compile → Run As → External Tools Configuration → Refresh Tab. Then check “Refresh resources upon completion” and select “The project containing the selected resource”.



Launching the application

After having compiled the project with `js_compile`, you can launch the application in the browser: Right click on the project → Run As → Run on server → Tomcat v6 → Finish.

The server starts and displays the application in the browser.



First Steps with Fybit

Defining the Program Entry Point

The JavaScript compiler needs to find the entry point to the application. This entry point is specified by the `@MainPanel` annotation, e.g. in the class

```
@MainPanel(name="index")
public class DemoPanel {
```

`DemoPanel.java`. The name attribute defines the name of the resulting JavaScript file. As a consequence the generated JavaScript file is referenced in the `index.html` with the "app-" prefix.

```
<script type="text/javascript" src="codebase/app-index.js"></script>
```

Remote Invocations

In order to make remote invocations to the server, you can use the `@Services` annotation to denote instances that call the server. There are two ways to do this:

```
// The TimeServiceProxy is created upon building the project
// with the 'compile' or 'js_compile' Ant target. Proxy classes allow
// to make asynchronous server calls
@Services(implementation=TimeServiceImpl.class, iface=TimeService.class)
protected static TimeServiceProxy service;

// For synchronous calls, the following declaration suffices:
@Services(implementation=TimeServiceImpl.class)
protected static TimeService service;
```

Rules for fields annotated with `@Services`:

- The field has to be `public static` or `protected static`.
- Use the interface as the type (here `TimeService`) and specify the concrete server class with the `implementation` attribute.
- If asynchronous calls to the server are needed, use the proxy class consisting of the interface's name and postfix "Proxy", e.g. `TimeServiceProxy`, and specify the interface with the `iface` attribute.

DOM Operations

Fybit allows DOM operations to be carried out in Java. To this end, the `DomUtil` provides useful methods. In the example application, a button is dynamically created and added to the page:

```
final DomUtil domUtil = Page.getInstance().getDomUtil();
HTMLButtonElement htmlButton = domUtil.createButton("asyncButton");
htmlButton.appendChild(domUtil.createText("Asynchronous Call"));
domUtil.getDivById("buttons").appendChild(htmlButton);
```

Using YUI Widgets

Riatrix builds on Yahoo!'s User Interface library (YUI)¹. Therefore, you can use all widgets that YUI provides. The classes are located in package `com.fybit.riatrix4js.yui`. Here some useful examples:

YUI Button with Click Listener

```
final Button asyncButton = Button.create("asyncButton");
asyncButton.addClickListener(new Listener() {
    public void perform () {
        // do something here
    }
});
```

Autocompletion with Server Calls

```
Function searchFct = Function.create(new Callback() {
    public Object executeCallback(Object[] args) {
        String input = (String)args[0];
        String[] result = myServer.get(input); // call server...
        return result;
    }
});
```

```
final DataSourceBase dataSource = FunctionDataSource.create(searchFct,
(FunctionDataSource.Conf)null);
dataSource.setMaxCacheEntries(100); // yes, we want to cache!
```

```
// input field:
AutoComplete ac = AutoComplete.create(
    "myField", "mySuggestions", dataSource, null
);
ac.setUseShadow(true);
```

The corresponding HTML part looks like this:

```
<div><input type="text" id="myField" name="myField"
style="width:220px;"><div id="mySuggestions"></div></div>
```

YUI Table with Data from existing HTML Table

```
Column.Conf[] myColumnDefs = new Column.Conf[] {
    Column.Conf.newKey("DATE").setLabel("Date"),
    Column.Conf.newKey("TITLE").setLabel("Book title"),
    Column.Conf.newKey("AUTHOR").setLabel("Author(s)")
};
```

```

DataSourceBase myDataSource =
DataSource.createDataSource(Dom.get("book_table"), null);

myDataSource.setResponseType(DataSourceBase.TYPE_HTMLTABLE());
myDataSource.setResponseSchema(ResponseSchema.Conf.newFields(new
ResponseField.Conf[] {
    ResponseField.Conf.newKey("DATE"),
    ResponseField.Conf.newKey("TITLE"),
    ResponseField.Conf.newKey("AUTHOR")
}));

DataTable myDataTable = DataTable.create("book_div",
    myColumnDefs,
    myDataSource,
    DataTable.Conf.newCaption("Enhance existing table"));

```

The HTML containing the table with the input data looks as follows:

```

<div id="book_div">
    <table id="book_table"><tbody>
        <tr>
            <td>1/23/1994</td>
            <td>Design Patterns</td>
            <td>Erich Gamma</td>
        </tr>
        ...
    </tbody></table>
</div>

```

Look at the example application for additional details and further examples.

Compile Options

The Ant target `js_compile` in `build.xml` allows you to set various compile options:

JavaScript Compression

```
<arg value="--compressjs"/>
```

Skip Specified YUI JavaScript Files

```
<arg value="--skipjsregex"/>
<arg value=".*(element|connection|get|json)\.js"/>
```

Skip Specified YUI CSS Files

```
<arg value="--skipcssregex"/>
<arg value=".*(reset-min|base-min|fonts-min|grids-min)\.css"/>
```

Examples

The example `DemoPanel.java` shows two simple features to get started: how to create an interactive table and how to react to events on a button. In `Showcase.java` more advanced features like local and remote auto-completion, tables, paginators, dialogs, menu, editors and buttons are shown.

ⁱ <http://developer.yahoo.com/yui/2/>