Jaffa Reference Guide

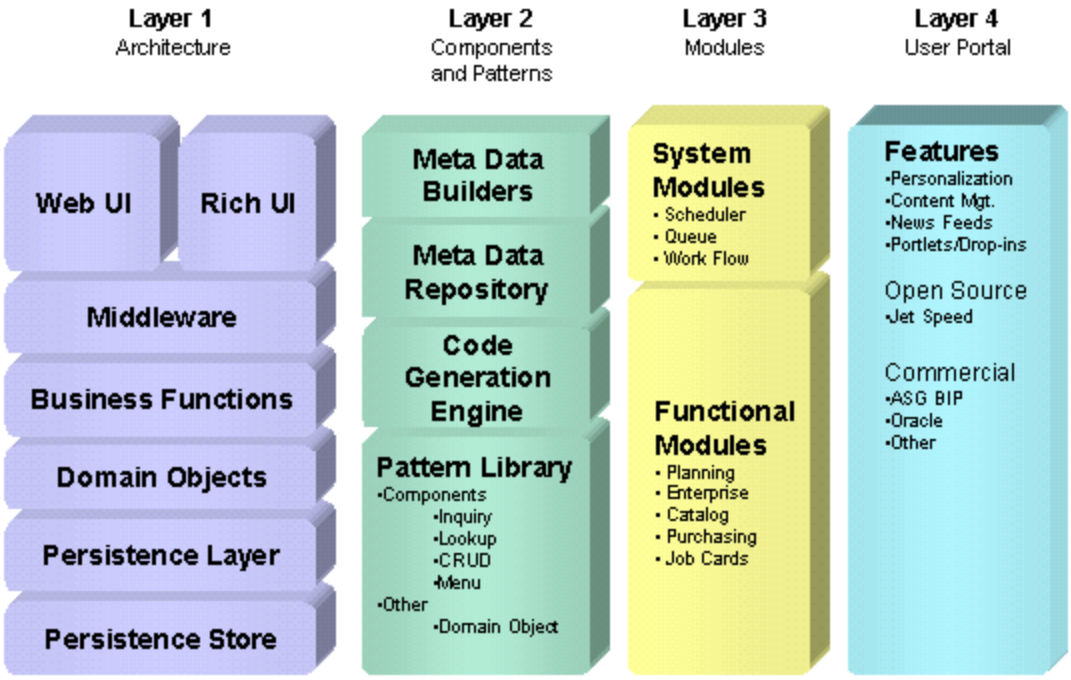
Jaffa Components

1 Contents

2 Introduction

The Jaffa component project is focused on delivering common business application components that are provide standard domain neutral functions (interfacing, messaging, alters, auditing, etc) the most enterprise applications require. We will also include in here components that enhance the Jaffa runtime framework to make it more useable. The first release of Jaffa Components is focused primarily on a set of admin and configuration tools specific to Jaffa, but it also includes a vanilla user account management component set.

This Components project is part of Jaffa's strategy for rabid business application development, as we move Jaffa beyond a basic runtime architecture, to include more generic functional modules.



(See - http://jaffa.sourceforge.net/documentation/)

As you can see from the architecture diagram above, the Jaffa Components project represents the Layer 3 System Modules.

3 Administration & Configuration

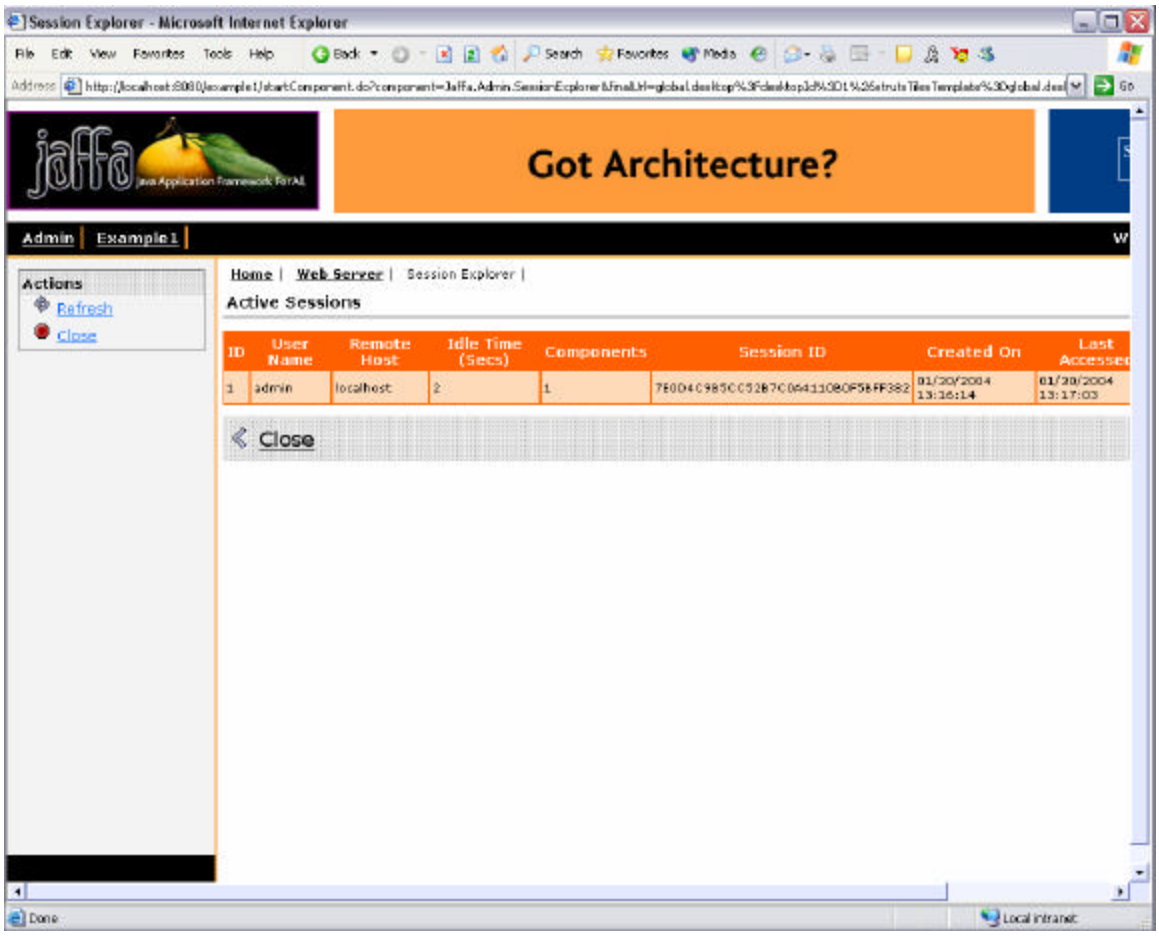
3.1 The Session Explorer

3.1.1 Overview

The Session explorer gives the ability to view all active user sessions, user object introspection and also component introspection.

3.1.2 How to Use

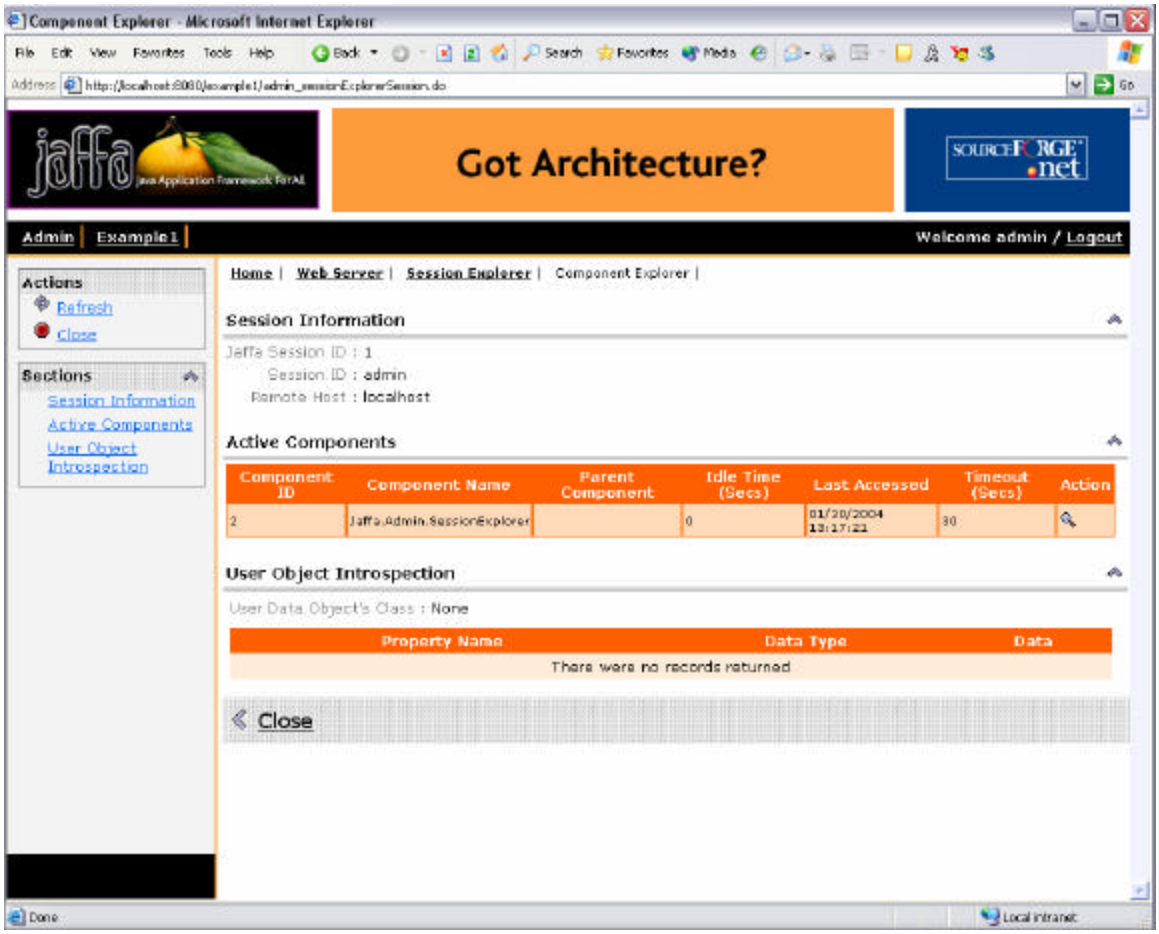
The Session Explorer is accessed by user with Admin or Developer role.
The Session Explorer is located under Admin | Web Server menu option.
The Session explorer main page list the active sessions with following fields



User Name - logged in user for that session
Remote host - Host name the session is logged in from.
Idle Time – How long the session has been idle (in seconds)
Components – No of components running for that session

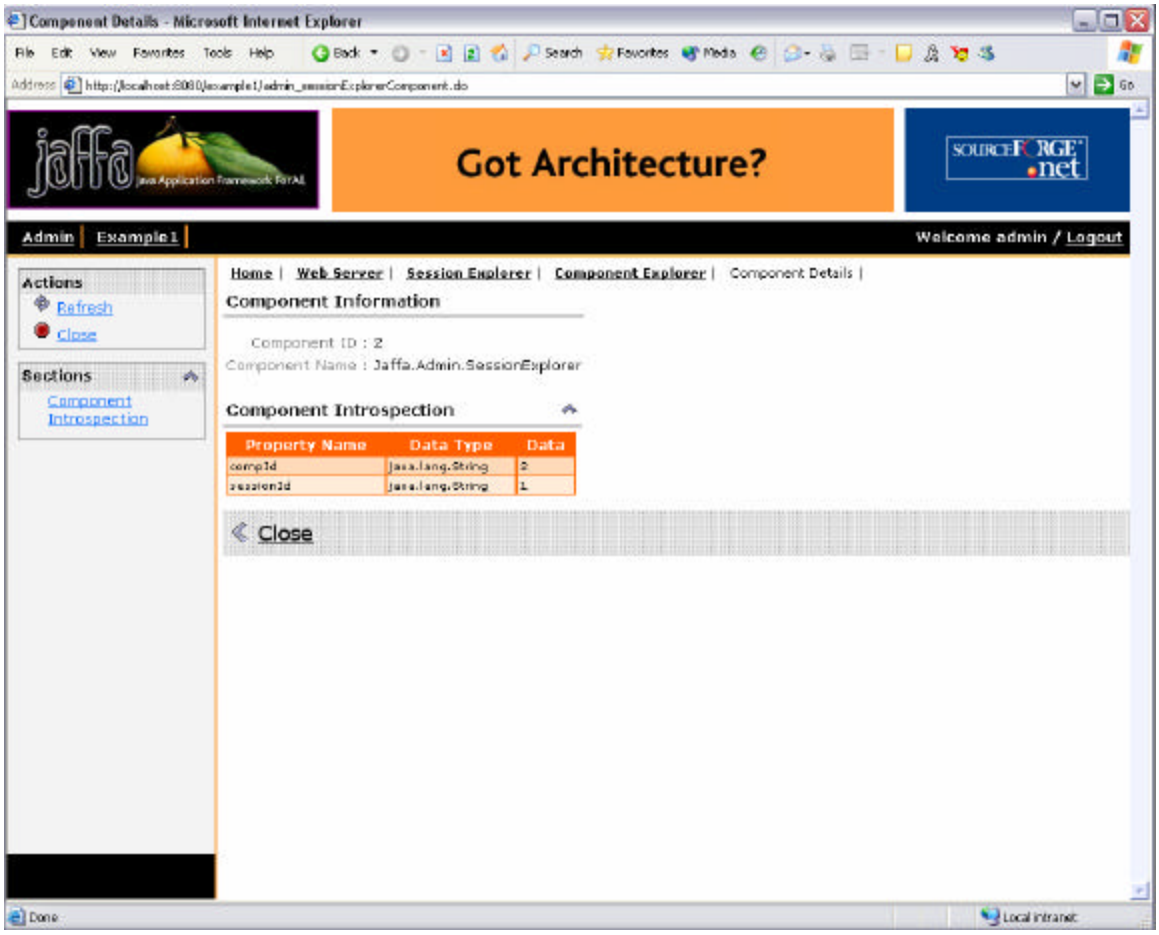
It also list the Session ID, Created on, Last Accessed and Timeout in seconds Once the session has timed out, the session is invalidated and the user will have to login to the system to access any component.

Clicking on View under Action column takes you to more details for that user session.



The Active components folding section lists the following fields
Component Id - Id assigned to the component
Component Name – Name of active Components
Parent Component – Parent Component, if any
Idle time – Time in seconds, the component has been idle.
Last Accessed - Last time the component was accessed by the user.
Timeout – Timeout in seconds. Once the Component is timed out that component is killed.

The Details/View action gives more details on that component.



The User Object Introspection folding section lists all the properties attached to the user session.

3.1.3 Setup and Configuration

No specific setup or configuration required for this component

3.2 The Log File Editor / Viewer

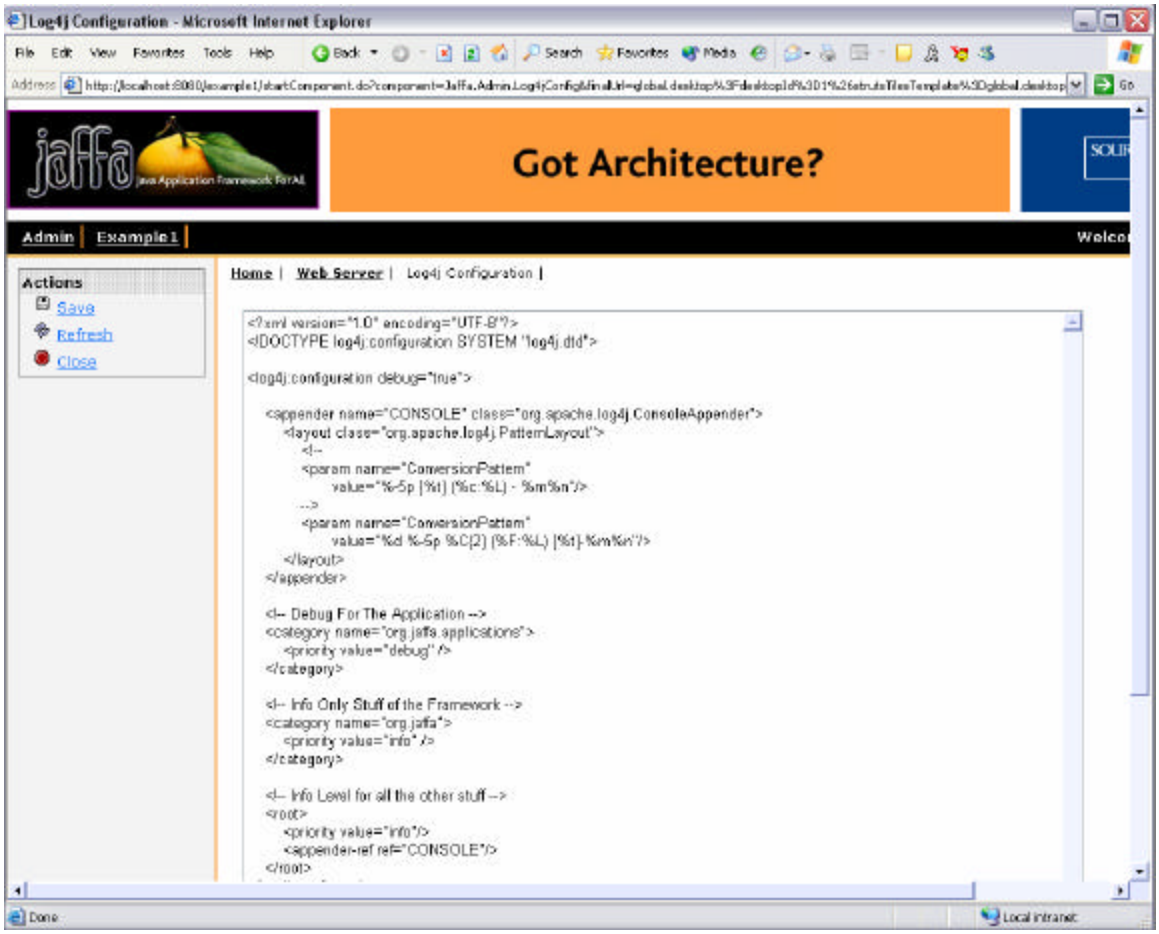
3.2.1 Overview

The Log file is used for configuring application logging. This file will only be used if specified by the framework.properties setting 'framework.Log4JConfig'. The setting should also provide the location of the file.

For details on Log4j visit http://logging.apache.org/log4j/docs/index.html
The Editor/Viewer gives the ability to configure the log4j.xml

3.2.2 How to Use

Log4j configuration can be accessed by user with System Admin role. On the main menu it can be accessed from the Admin | Web Server. The main page of the Log4j configuration component displays the contents of the log4j.xml file. The Save action will validate the xml structure against the dtd and if valid will save the contents to the log4j.xml file.
Refresh will refresh the screen with the latest contents of the xml file.
Close will close the component.



3.2.3 Setup and Configuration

As mentioned earlier, to use the log4j.xml file for logging it needs to be specified in the `framework.properties` file.

- For no configuration on startup, set to 'none'. Only use this setting if the Web Container has already set up log4j.
  Example: `framework.Log4JConfig=none`

- For a basic configurator (which logs all messages to the console), set the value 'default'. Example: `framework.Log4JConfig=default`
- For specific setting, specify the location of a Log4J XML Config File relative to the classpath.
  A separate thread will be created to monitor the Log4J XML Config File, every 60 seconds, for any change.
  Example: `framework.Log4JConfig=resources/log4j-config.xml`

```
# Log4j Configuration To Use.
#    For no configuration on startup, set to 'none'. Only use this setting if the
Web Conainter has already set up log4j
#    Example: framework.Log4JConfig=none
#    For a basic configurator (which logs all messages to the console), set the
value 'default'.
#    Example: framework.Log4JConfig=default
#    For specific setting, specify the location of a Log4J XML Config File relative
to the classpath.
#    A separate thread will be created to monitor the Log4J XML Config File, every
60 seconds, for any change.
#    Example: framework.Log4JConfig=log4j-config.xml
framework.Log4JConfig=log4j-config.xml
```

3.3 The Labels Editor

3.3.1 Overview

A Jaffa application obtains all labels (and titles and error messages) from a resource bundle. For the purpose of this document, any reference to a label is applicable to titles as well as error messages that are obtained from the resource bundle.

To manage the huge number of labels, the source code typically consists of fragment files with default values for the labels. At build time, the fragment files are merged into one big resource file to be deployed into the web application.

The LabelEditor component can be used by an end-user to override the default value for any of the labels.

This component can also be used by a developer to sync the source fragment files with the override values.

The LabelEditor component depends on 3 resource files:

- `ApplicationResources.default` – This file contains the default values for the labels. It is built by merging all the source fragment files.

- `ApplicationResources.override` – This file will store all the override values. It should ideally be placed outside the web application, so that it can survive all the product upgrades.
- `ApplicationResources.properties` – The web application obtains the labels from this file. The InitServlet creates this file by merging the default and override files on startup.

For the purpose of this document, we'll refer to the 3 files as default, override and properties files respectively.
The LabelEditor component will not work in the absence of the above files.

3.3.2 How to Use

The LabelEditor component can be invoked by the URL

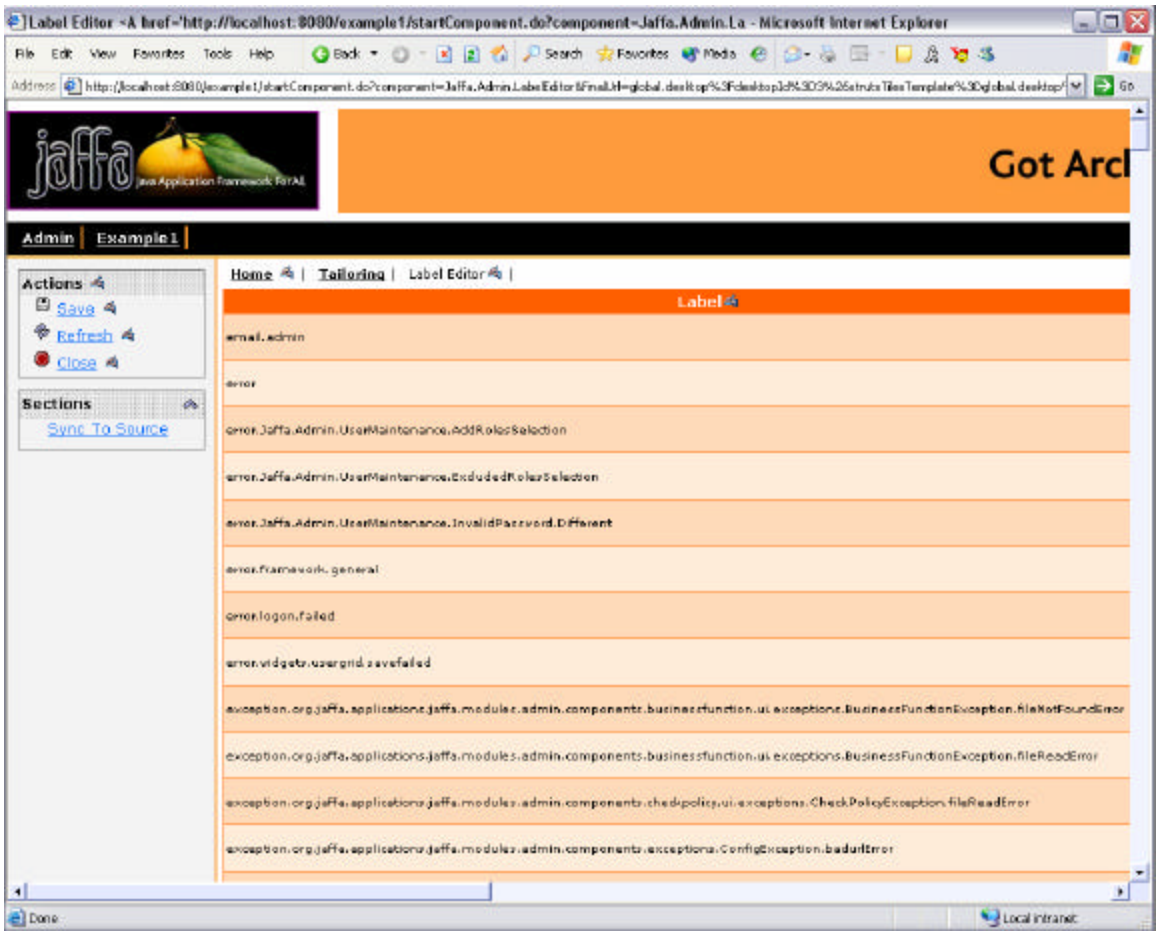
```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.LabelEditor
```

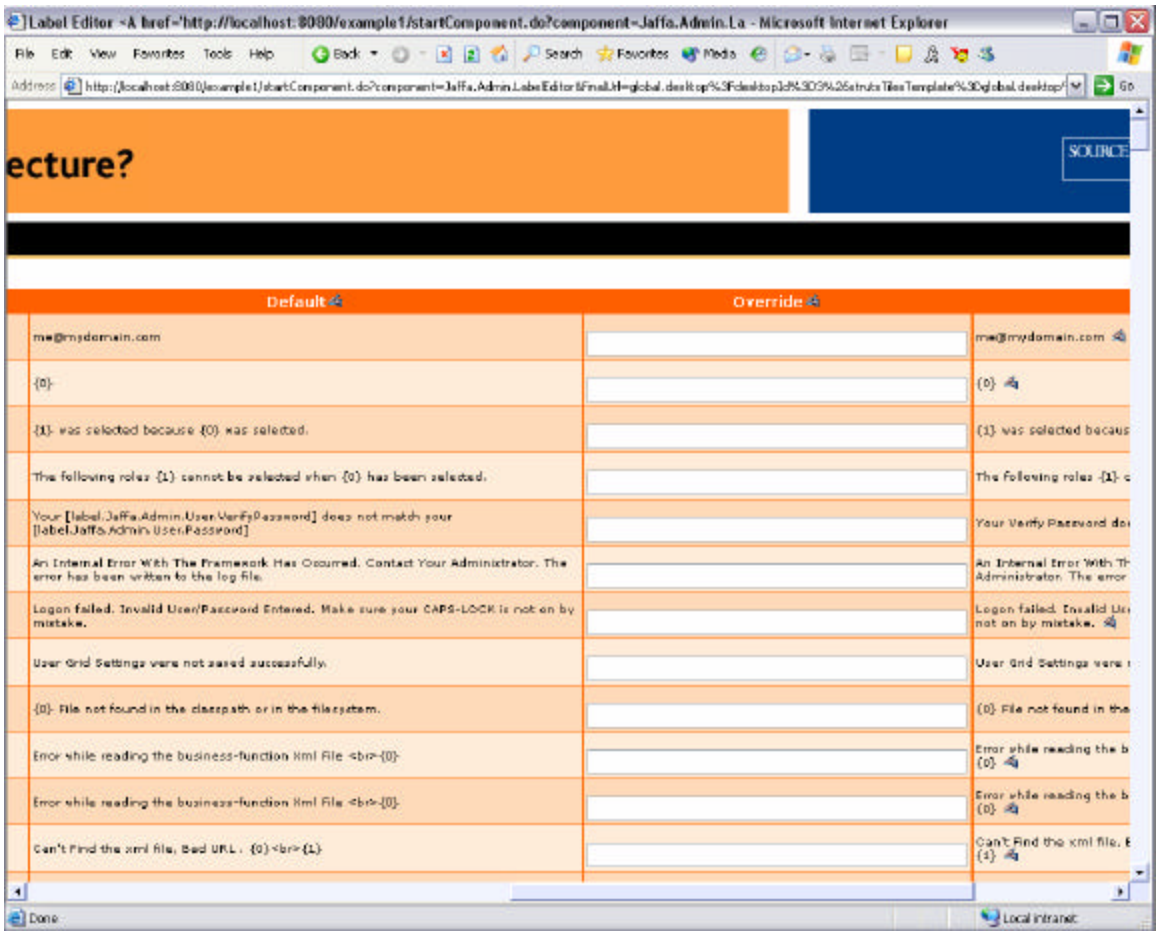
When running the Jaffa baseline application, it can be accessed from

```
Admin -> Tailoring -> Labels
```

The LabelEditor component displays all the entries from the default file.
For each entry, the default value, the override value (if any) and the display value will be displayed.
Pass the '`labelFilter`' parameter in the URL, if you want to retrieve only one entry.
For eg:

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.LabelEditor&labelFilter=Xyz
```


Label Editor <A href='http://localhost:8080/example1/startComponent.do?component=Jaffa.Admin.La - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Got Arcl
Admin
Example1
Actions
Save
Refresh
Close
Sections
Sync To Source
Home | Tailoring | Label Editor |
Label
email.admin
error
error.Jaffa.Admin.UserMaintenance.AddRolesSelection
error.Jaffa.Admin.UserMaintenance.ExcludedRolesSelection
error.Jaffa.Admin.UserMaintenance.InvalidPassword.Different
error.framework.general
error.logon.failed
error.widgets.usergrid.savefailed
exception.org.jaffa.applications.jaffa.modules.admin.components.businessfunction.ui.exceptions.BusinessFunctionException.fileNotFoundError
exception.org.jaffa.applications.jaffa.modules.admin.components.businessfunction.ui.exceptions.BusinessFunctionException.fileReadError
exception.org.jaffa.applications.jaffa.modules.admin.components.checkpolicy.ui.exceptions.CheckPolicyException.fileReadError
exception.org.jaffa.applications.jaffa.modules.admin.components.exceptions.ConfigException.badUrlError
Done
Local intranet



A user, having access to the business function '`Jaffa.Admin.Labels`', can add/update/delete an override value for one or more entries.
Click the 'Save' button to store the changes in the override file.
These changes will be immediately visible. Just bring up a screen containing the changed label.

Hit the 'Refresh' button at any time to get back the last stored values.

A developer, having access to the business function '`Jaffa.Admin.Labels.Developer`', can sync the source fragment files with the override values. This requires the following 2 values

- `fragmentName` - The fragment file which will contain the original properties.
- `searchPath` - The directory(s), which will be recursively searched, for the source fragment files.

Click the 'SyncToSource' button to transfer all the override values to the appropriate source fragment file. The override file will be blanked out by the sync process.

The Jaffa widgets - Label, UserGridColumn, Button, FoldingSection, and the HistoryNav and the Error Popup generate links to the LabelEditor component, provided the user has access to the component. This link will allow a user to modify any label.

3.3.3 Setup and Configuration

A Jaffa application needs the name and locations for the default, override and properties files. This information is provided by the 3 parameters that can be passed to the `InitServlet` in web application descriptor 'web.xml' file.

```
  <servlet>
    <servlet-name>InitApp</servlet-name>
    <servlet-class>org.jaffa.config.InitApp</servlet-class>
    <init-param>
      <!-- This file will be picked up from WEB-INF/classes, i.e relative to the
classpath -->
      <param-name>framework.ApplicationResourcesLocation</param-name>
      <param-value>ApplicationResources.properties</param-value>
    </init-param>
    <init-param>
      <!-- This file will be picked up from WEB-INF/classes, i.e relative to the
classpath -->
      <param-name>framework.ApplicationResourcesDefaultLocation</param-name>
      <param-value>ApplicationResources.default</param-value>
    </init-param>
    <init-param>
      <!-- This file will be picked up using File I/O -->
      <param-name>framework.ApplicationResourcesOverrideLocation</param-name>
      <param-value>C:/ApplicationResources.override</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

Note: The value for the '`framework.ApplicationResourcesLocation`' parameter, as specified above, should be consistent with the value passed for the message-resources to Struts.

To instantly apply the label changes to the web application, configure Struts to use a custom Jaffa `MessageResourcesFactory`.

```
<!-- The following parameters can be passed to the Struts servlet in web.xml -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>factory</param-name>
      <param-value>org.jaffa.util.PropertyMessageResourcesFactory</param-value>
    </init-param>
    . . .
  </servlet>
```

```
<!-- Alternately , the following definition can be passed in struts-config.xml -->
<message-resources parameter="ApplicationResources"
factory="org.jaffa.util.PropertyMessageResourcesFactory"/>
```

Note: If the Jaffa `MessageResourcesFactory` is not used, then the web application will have to reloaded to view the changes to the labels.

The following are the business functions used by the '`Jaffa.Admin.LabelEditor`' component

- `Jaffa.Admin.Labels`: This is a mandatory function for the LabelEditor component
- `Jaffa.Admin.Labels.Developer`: This is an optional function for the LabelEditor component which allows access to the Sync feature

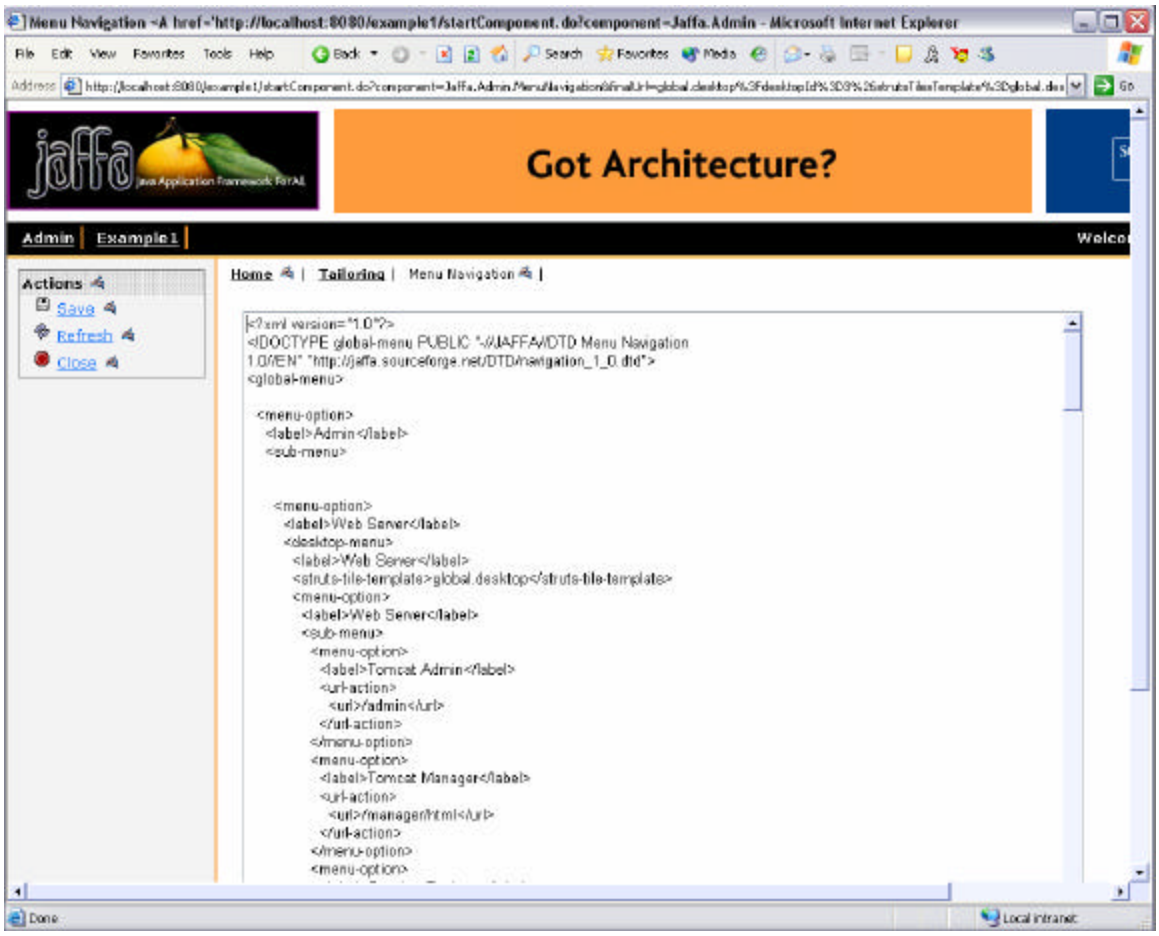
3.4 The Navigation Editor

3.4.1 Overview

The Navigation editor allows you to view/edit the navigation.xml at runtime. This file controls what is displayed on the menu.

3.4.2 How to Use

The Navigation editor is accessed by the user having System Admin role. On the main menu the component is located under Admin | Tailoring. The main page of the Navigation component displays the contents of the navigation.xml file.



The Save action validates the xml structure against the dtd and if valid, saves the contents to the navigation.xml.
Note: The Menu will be refreshed on save only for the logged in user. For all other logged on user the changes will take effect on the next login.
The Refresh action re-retrieves the contents of navigation.xml file
The Close action closes the component.

3.4.3 Setup and Configuration

The location of the navigation.xml file is specified in `framework.properties`.
If the value is not specified it looks in the classpath for '`resources/roles.xml`'

```
# Location to load the menu layout file from
# In Jaffa v1.x this defaults to classpath://resources/menu-list.xml for the v1.x
menu system
# In Jaffa v2.x this defaults to classpath://resources/navigation.xml for the new
v2.x global menu / desktop system
# framework.menu.url=classpath:///resources/navigation.xml
```

3.5 The Validation Rules Editor

3.5.1 Overview

The ValidationRulesEditor component can be used for editing the configuration files for the Rules Engine.

The configuration of the Rules Engine is based on the following files

- A core rules file. This file contains the list of validations to be applied for a domain-field
- An optional variant rules file. This file will extend or override the core validations for a domain-field
- One or more validator files. This file will contain the definitions for the different validators, i.e the Java class to be used for applying a validation.

3.5.2 How to Use

The ValidationRulesEditor component can be invoked by the URL

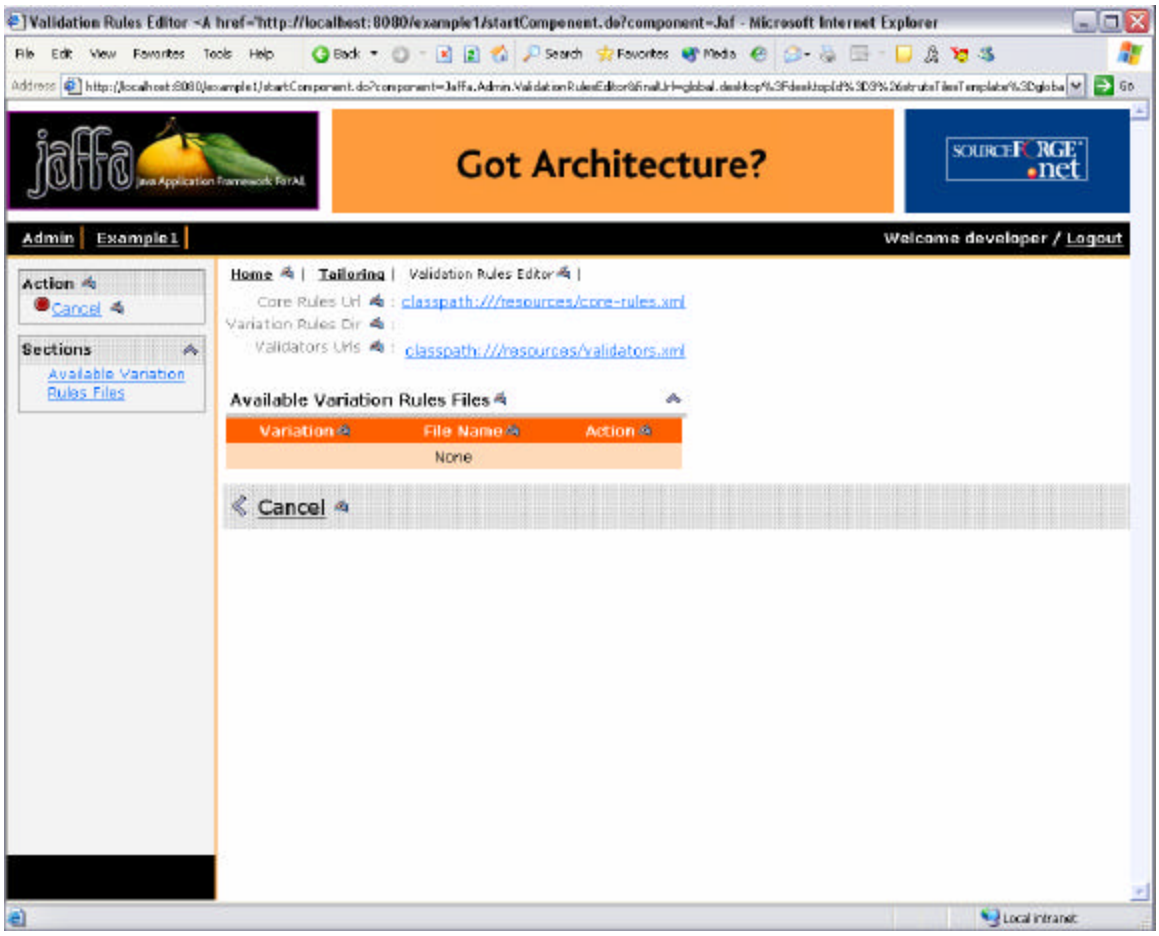
```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.ValidationRules
Editor
```

When running the Jaffa baseline application, it can be accessed from

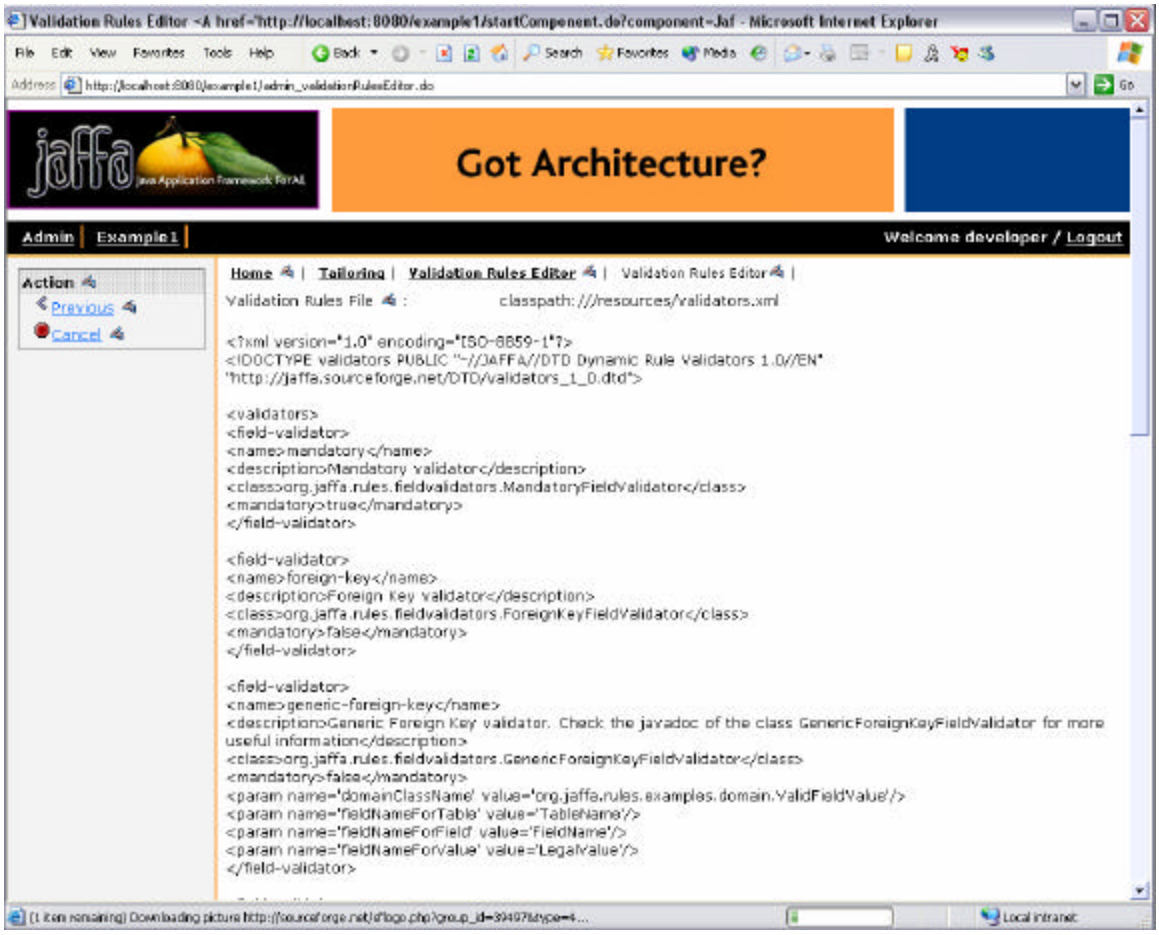
```
Admin -> Tailoring -> Validation Rules
```

The component displays the following files

- The core rules file
- One or more validator rules files
- One or more variant rules files


Validation Rules Editor -A href="http://localhost:8080/example1/startCompenent.de?component=Jaf - Microsoft Internet Explorer
Got Architecture?
Admin
Example1
Welcome developer / Logout
Action
Cancel
Sections
Available Variation Rules Files
Home | Tailoring | Validation Rules Editor |
Core Rules Url : classpath:///resources/core-rules.xml
Variation Rules Dir :
Validators Urls : classpath:///resources/validators.xml
Available Variation Rules Files
Variation
File Name
Action
None
Cancel

Click a file to bring up its contents.



Click the ‘Save’ button, after editing the file, to store the changes. Changes will be applied instantly to the application.
Click the ‘Refresh’ button to obtain the last saved contents of the file.

NOTE: A configuration file inside a JAR will be view-only and cannot be modified.

3.5.3 Setup and Configuration

The configuration files are specified through the framework properties

```
# Specify the following properties in org/jaffa/config/framework.properties file
for configuring the Rules Engine

# This property holds the url for the core-rules file used by the Dynamic Rules
Engine.
# The default-value is 'classpath:///resources/core-rules.xml'
framework.rules.core-rules.url=classpath:///resources/core-rules.xml
```

```
# This property holds the directory in which the customer variations to the core-
rules for the Dynamic Rules Engine will be located.
# This property may be left empty, if no variations are being used.
# Eg: 'file:///C:/sandbox/custom-rules-dir'
# NOTE: The variation files, if any, should be named as <variation>-rules.xml
framework.rules.variations.directory=

# This property holds the comma-separated list of the various validator.xml urls
used by the Dynamic Rules Engine.
# The default-value is 'classpath:///resources/validators.xml'
# Eg: 'classpath:///resources/validators.xml,file:///C:/sandbox/validators/custom-
validator.xml'
framework.rules.validators.url.list=classpath:///resources/validators.xml
```

The following business function is used by the '`Jaffa.Admin.ValidationRulesEditor` component

- `Jaffa.Admin.EditValidationRules`: This is a mandatory function for the ValidationRulesEditor component

3.6 The Default Value Editor

3.6.1 Overview

A maintenance component in Jaffa obtains the default values for its fields from the optional '`ComponentDefaultValues.properties`' file located in the base package for the component. This file follows the rules of a Java Properties file. It should contain field/value pairs.

The DefaultValueEditor component is used for editing these files.

3.6.2 How to Use

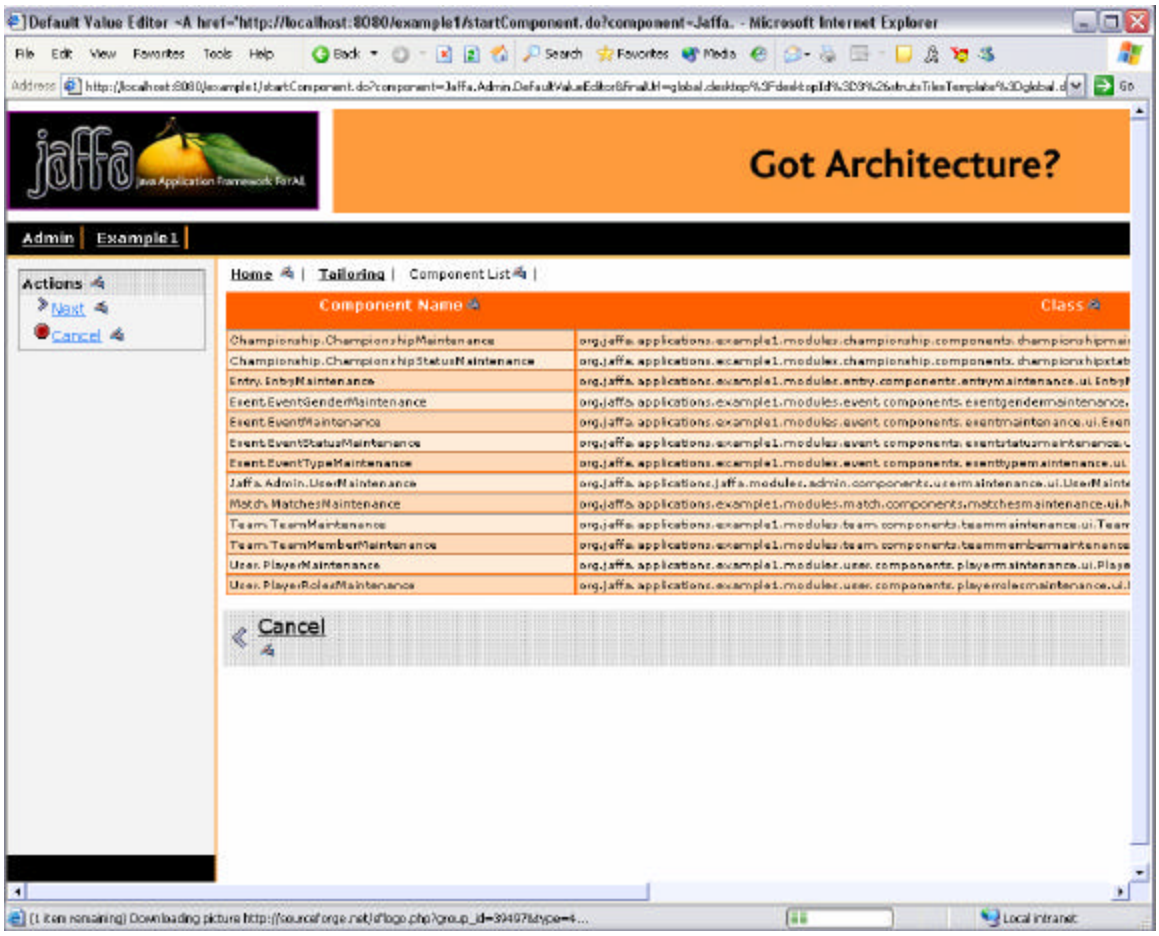
The DefaultValueEditor component can be invoked by the URL

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.DefaultValueEdi
tor
```

When running the Jaffa baseline application, it can be accessed from

```
Admin -> Tailoring -> Default Values
```

The component lists all the components which have the '`ComponentDefaultValues.properties`' file in their base package.



Double-click on any row to view the contents of the file for a component.
Click the 'Save' button, after editing the file, to store the changes.
Changes will be applied instantly to the application.

Click the 'Refresh' button to obtain the last saved contents of the file.

NOTE: A '`ComponentDefaultValues.properties`' file inside a JAR will be view-only and cannot be modified.

3.6.3 Setup and Configuration

The following business function is used by the '`Jaffa.Admin.DefaultValueEditor`' component

- `Jaffa.Admin.EditDefaultValue`: This is a mandatory function for the DefaultValueEditor component

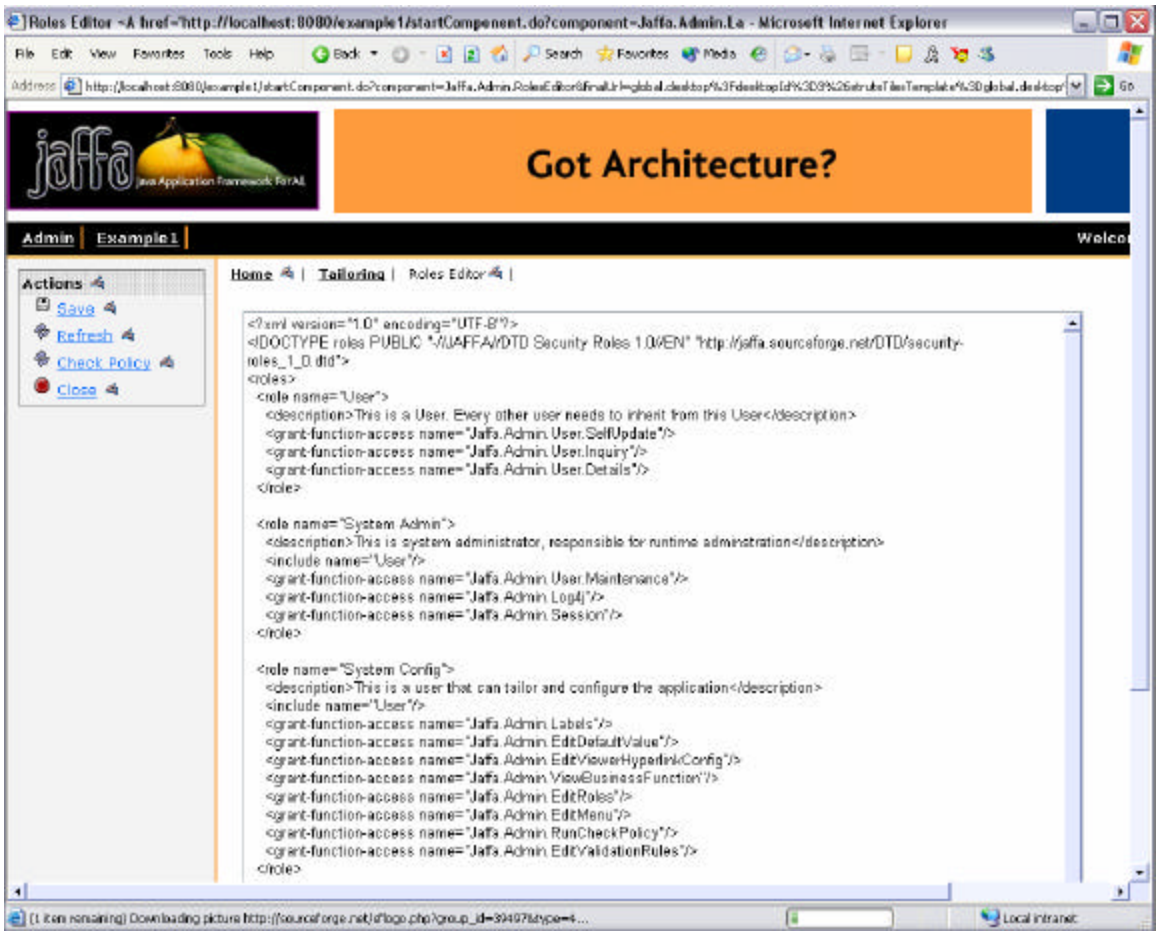
3.7 The Roles Editor

3.7.1 Overview

The Roles Editor allows you to view/edit the roles.xml .The Editor also allows you to perform semantic validation of the roles.xml cross-referencing with the business-function.xml for valid business-functions

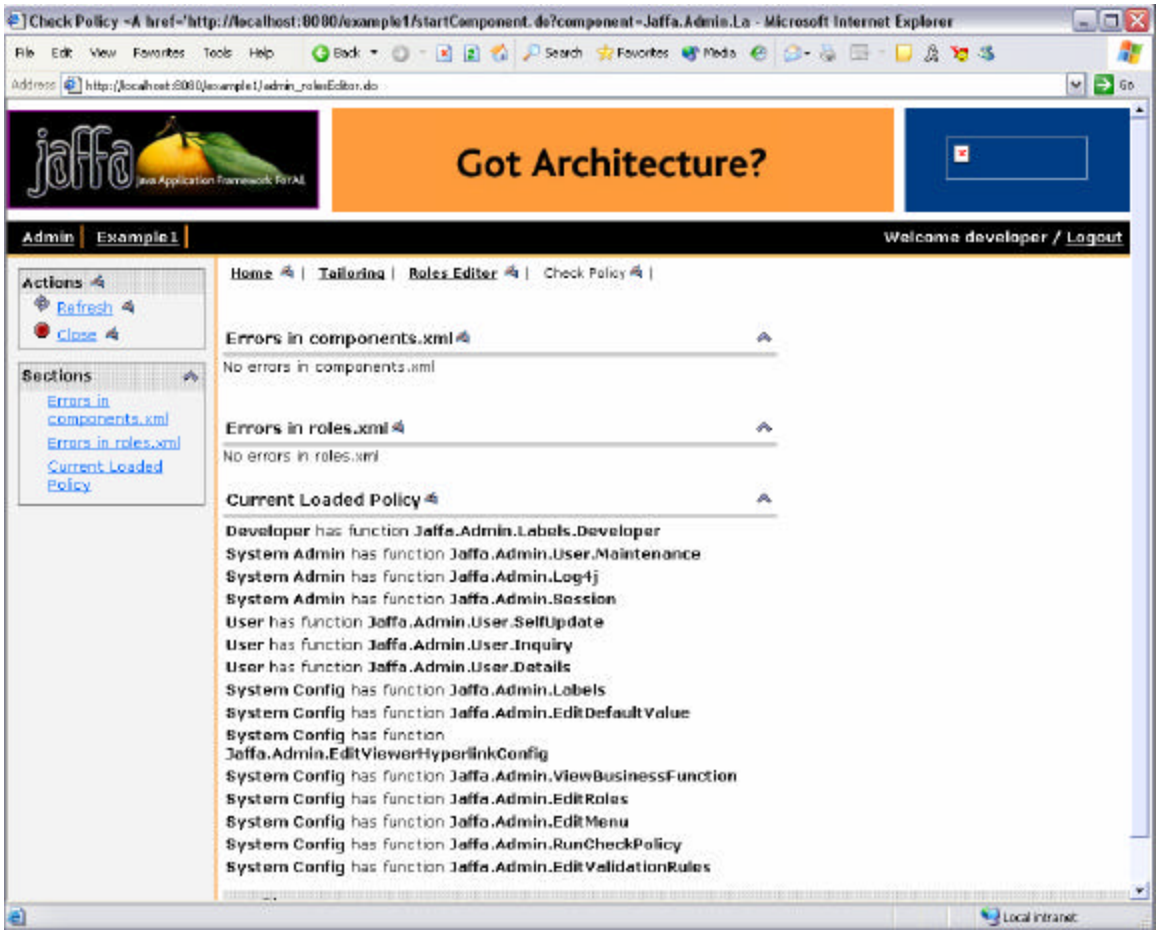
3.7.2 How to Use

The Roles Editor is accessed by user who has System Admin role. On the main menu it is located under Admin | Tailoring. The main page displays the contents of the roles.xml file in the Update mode.



The Save Action will save the contents to the roles.xml file and clears the cache .
The Refresh Action will refresh the screen with the latest contents of the roles.xml file
Check Policy Action will run the Check Policy component which will look for

- Report errors in roles.xml for any invalid business-functions
- Report errors in components.xml for any invalid business-functions
- List the currently loaded roles and business-function mapping.



3.7.3 Setup and Configuration

The location of the roles.xml file is specified in framework.properties.
If the value is not specified it looks in the classpath for ‘resources/roles.xml’

```
# Location to load the role based secuirty policy file from
# If no value is specified, it looks in the classpath for 'resources/roles.xml'
#framework.security.policy.url=classpath:///resources/roles.xml
```

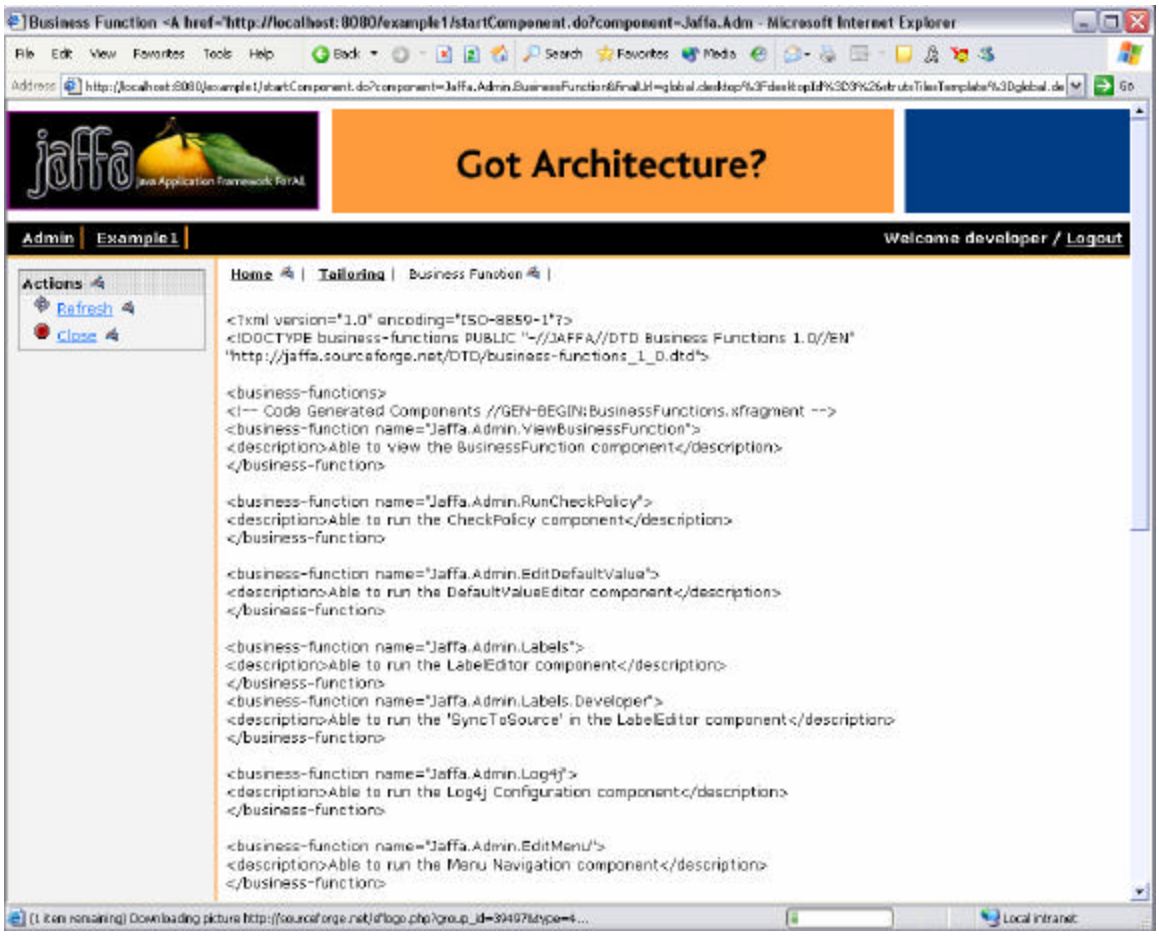
3.7.4 The Business Function Editor

3.7.5 Overview

The Business Function Editor allows you to view/edit the business-functions.xml .The Editor also allows you to view the contents of business-functions.xml
This is a read only feature and does not allow editing the business-functions.

3.7.6 How to Use

The Business Function Editor is accessed by user who has System Admin role. On the main menu it is located under Admin | Tailoring. The main page displays the contents of the business-functions.xml file in Read only mode.



The Refresh Action will refresh the screen with the latest contents of the business-functions.xml file

3.7.7 Setup and Configuration

The editor looks for resources/business-function.xml. There is no property for the location in framework.properties file.

3.8 The Viewer Hyperlink Config Editor

3.8.1 Overview

The Jaffa Text widget can generate hyperlinks to appropriate Viewer components based on the ViewerHyperlink configuration.

The configuration is based on 2 files

- DomainFieldViewerComponentMappingFile – This maps a domain field to a Viewer component
- KeyFieldPerViewerComponentFile – This lists the key-field for a Viewer component

So if the 'domain' and 'domainField' attributes are passed to a Text widget, then the widget will look up the 2 configuration files and generate a hyperlink to the appropriate Viewer component.

```
<%-- Example of using a Text widget in a JSP --%>

<Portlet:Text
domain='org.example.applications.application1.modules.module1.domain.SomeDomain'
domainField='SomeField'/>
```

This gives the application a very consistent feel.

3.8.2 How to Use

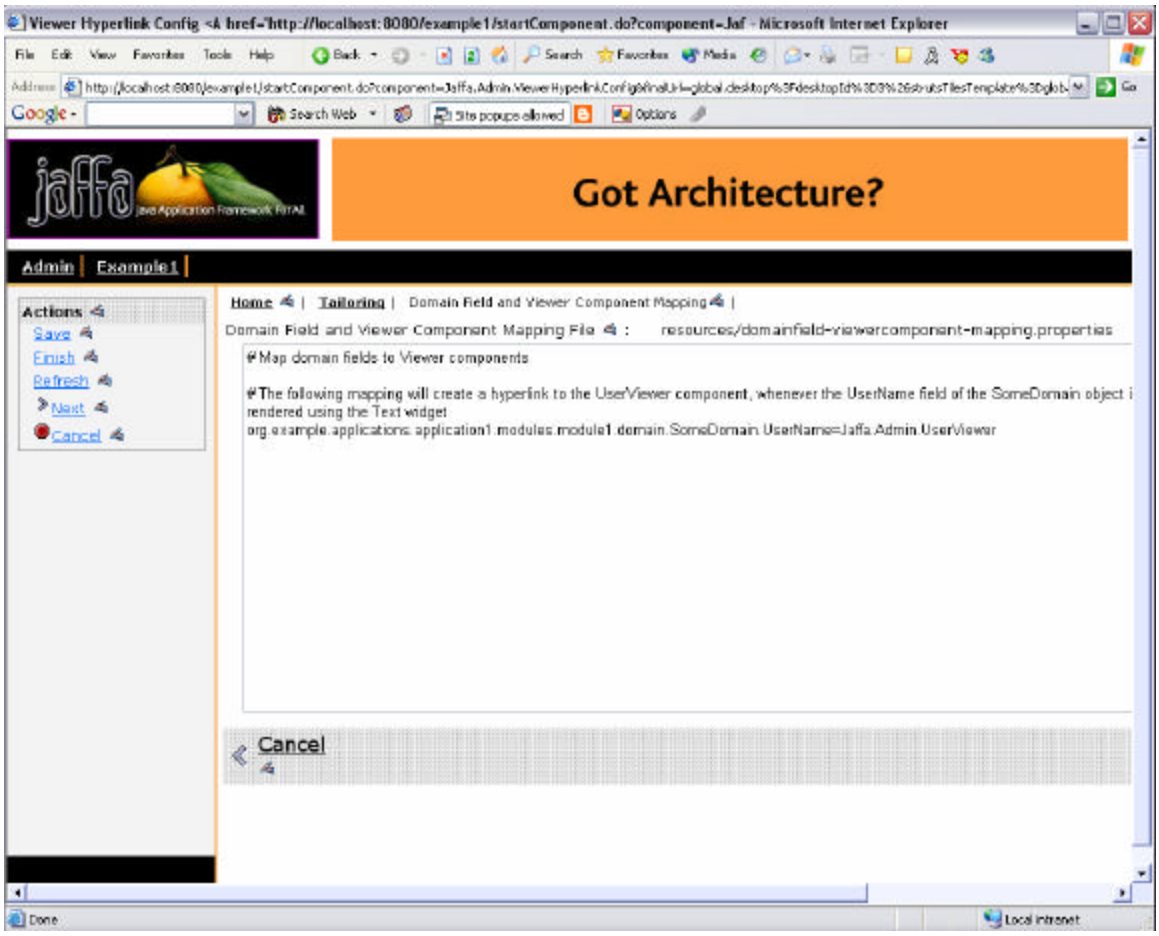
The ViewerHyperlinkConfig component can be invoked by the URL

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.ViewerHyperlink
Config
```

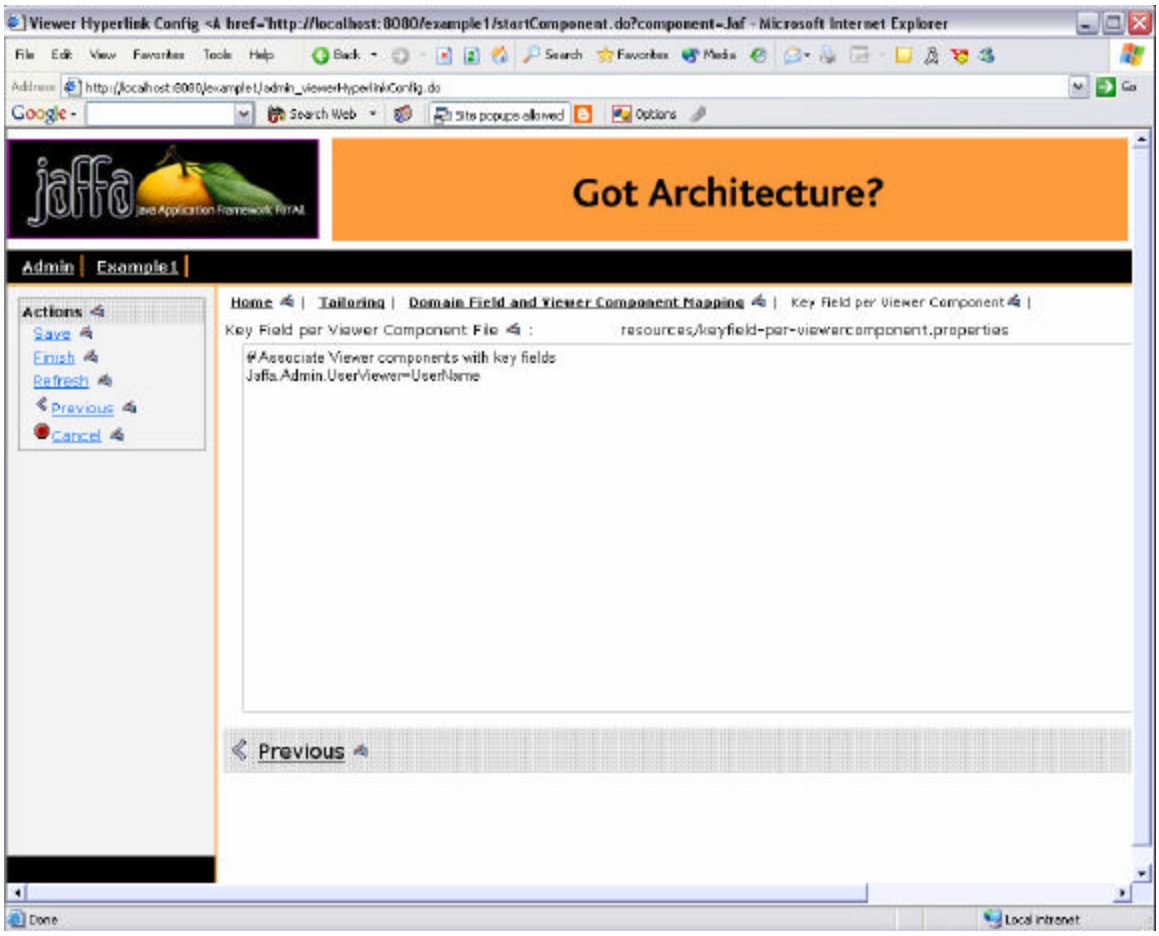
When running the Jaffa baseline application, it can be accessed from

```
Admin -> Tailoring -> Viewer Hyperlinks
```

The component initially displays the DomainFieldViewerComponentMappingFile.


Viewer Hyperlink Config <A href="http://localhost:8080/example1/startComponent.do?component=Jaf - Microsoft Internet Explorer
Got Architecture?
Admin
Example1
Actions
Save
Finish
Refresh
Next
Cancel
Home | Tailoring | Domain Field and Viewer Component Mapping |
Domain Field and Viewer Component Mapping File : resources/domainfield-viewercomponent-mapping.properties
#Map domain fields to Viewer components
#The following mapping will create a hyperlink to the UserViewer component, whenever the UserName field of the SomeDomain object i
rendered using the Text widget
org.example.applications.application1.modules.module1.domain.SomeDomain.UserName=Jaffa.Admin.UserViewer
Cancel
Done
Local intranet

Click the 'Next' button to display the KeyFieldPerViewerComponentFile.



On each screen:
Click the 'Save' button, after editing the file, to store the changes. Changes will be applied instantly to the application.
Click the 'Refresh' button to obtain the last saved contents of the file.

NOTE: A configuration file inside a JAR will be view-only and cannot be modified.

3.8.3 Setup and Configuration

The 2 configuration files are specified through the framework properties

```
# Specify the following 2 properties in org/jaffa/config/framework.properties file
to turn on the Viewer Hyperlink feature

# The TextTag uses this properties file to determine if a hyperlink to a Viewer
component should be generated for a field
```

```
# Absence of this property or a blank value will disable the hyperlink feature in
TextTag
# The value for this property, if any, should point a properties file relative to
the classpath
# Eg. 'resources/domainfield-viewercomponent-mapping.properties'
framework.DomainFieldViewerComponentMappingFile=resources/domainfield-
viewercomponent-mapping.properties

# The TextTag uses this properties file to determine the key-field of a Viewer
component for which it has generated a hyperlink
# Absence of this property or a blank value will disable the hyperlink feature in
TextTag
# The value for this property, if any, should point a properties file relative to
the classpath
# Eg. 'resources/keyfield-per-viewercomponent.properties'
framework.KeyFieldPerViewerComponentFile=resources/keyfield-per-
viewercomponent.properties
```

The following is a sample of the DomainFieldViewerComponentMappingFile

```
# Map domain fields to Viewer components

# The following mapping will create a hyperlink to the UserViewer component,
whenever the UserName field of the SomeDomain object is rendered using the Text
widget
org.example.applications.application1.modules.module1.domain.SomeDomain.UserName=Ja
ffa.Admin.UserViewer

. . .
```

The following is a sample of the KeyFieldPerViewerComponentFile

```
# Associate Viewer components with key fields
Jaffa.Admin.UserViewer=UserName

. . .
```

To turn off this feature completely, simply blank out all the entries in the 2 configuration files. Alternately, do not specify any values for the 2 framework properties.

To disable this feature for a particular Text widget, pass a ‘true’ value for the attribute ‘disableHyperlink’.

```
<%-- Example of using a Text widget in a JSP with the ViewerHyperlink turned off --
%>

<Portlet:Text
domain='org.example.applications.application1.modules.module1.domain.SomeDomain'
domainField='SomeField' disableHyperlink='false'/>
```

The following business function is used by the
'Jaffa.Admin.ViewerHyperlinkConfig component

- Jaffa.Admin.EditViewerHyperlinkConfig: This is a mandatory function for the ViewerHyperlinkConfig component

3.9 User Account Management

3.9.1 Overview

The following components are available for account management

- Jaffa.Admin.UserMaintenance: Use this for creating/updating/deleting user accounts
- Jaffa.Admin.UserFinder: This can be used for searching user accounts. It has links for creating/updating/deleting user accounts
- Jaffa.Admin.UserSelfUpdate: This component is available for a user, who does not have access to the Jaffa.Admin.UserMaintenance component, to update personal information like password, email.
- Jaffa.Admin.UserViewer: Use this for viewing a user account. It has a link for updating the user account.
- Jaffa.Admin.UserLookup: This is the Loookup component for user accounts

3.9.2 How to Use

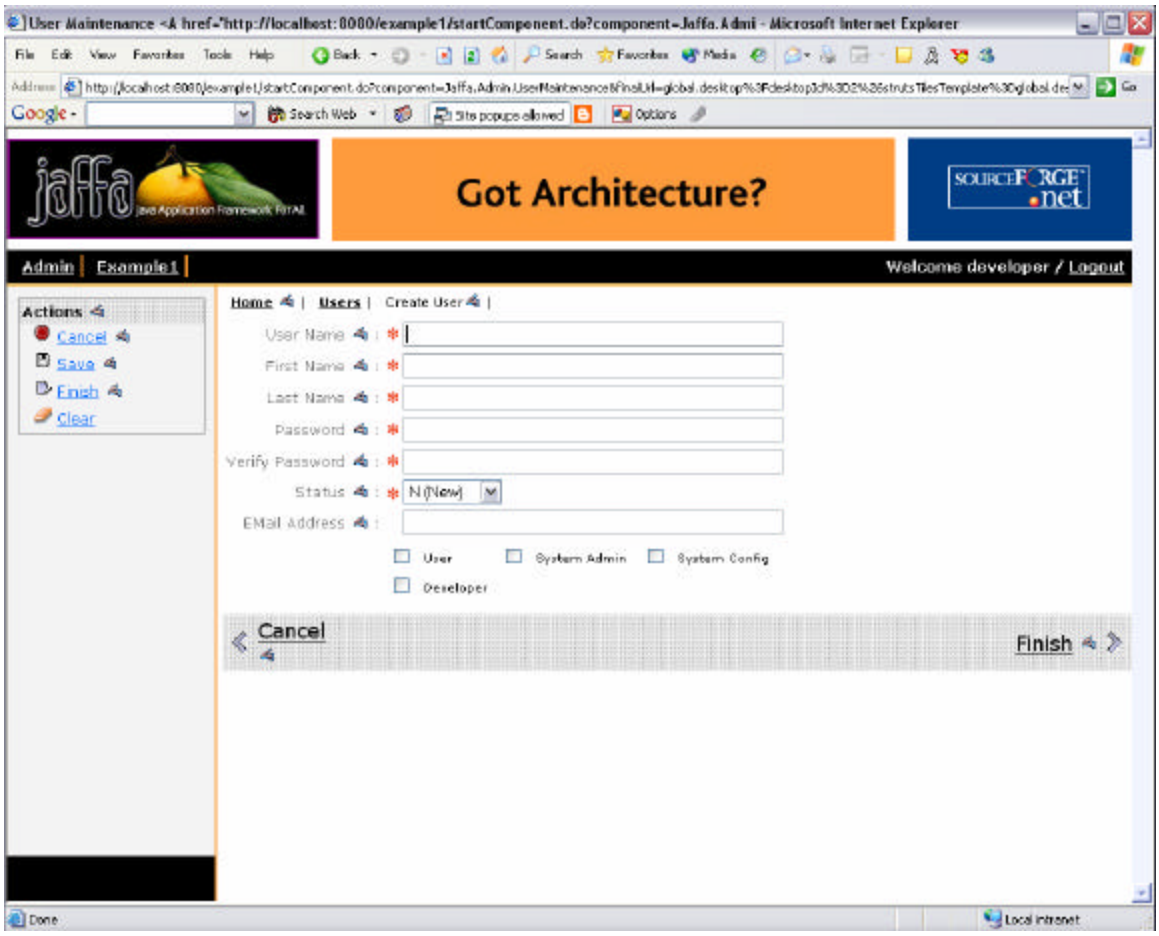
3.9.2.1 Creating a new user account

The UserMaintenance component can be invoked by the URL

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.UserMaintenance
```

When running the Jaffa baseline application, it can be accessed from.

```
Admin -> Users -> Create New User
```



Enter all the mandatory fields and hit ‘Save’ to create a user account.

3.9.2.2 Updating a user account

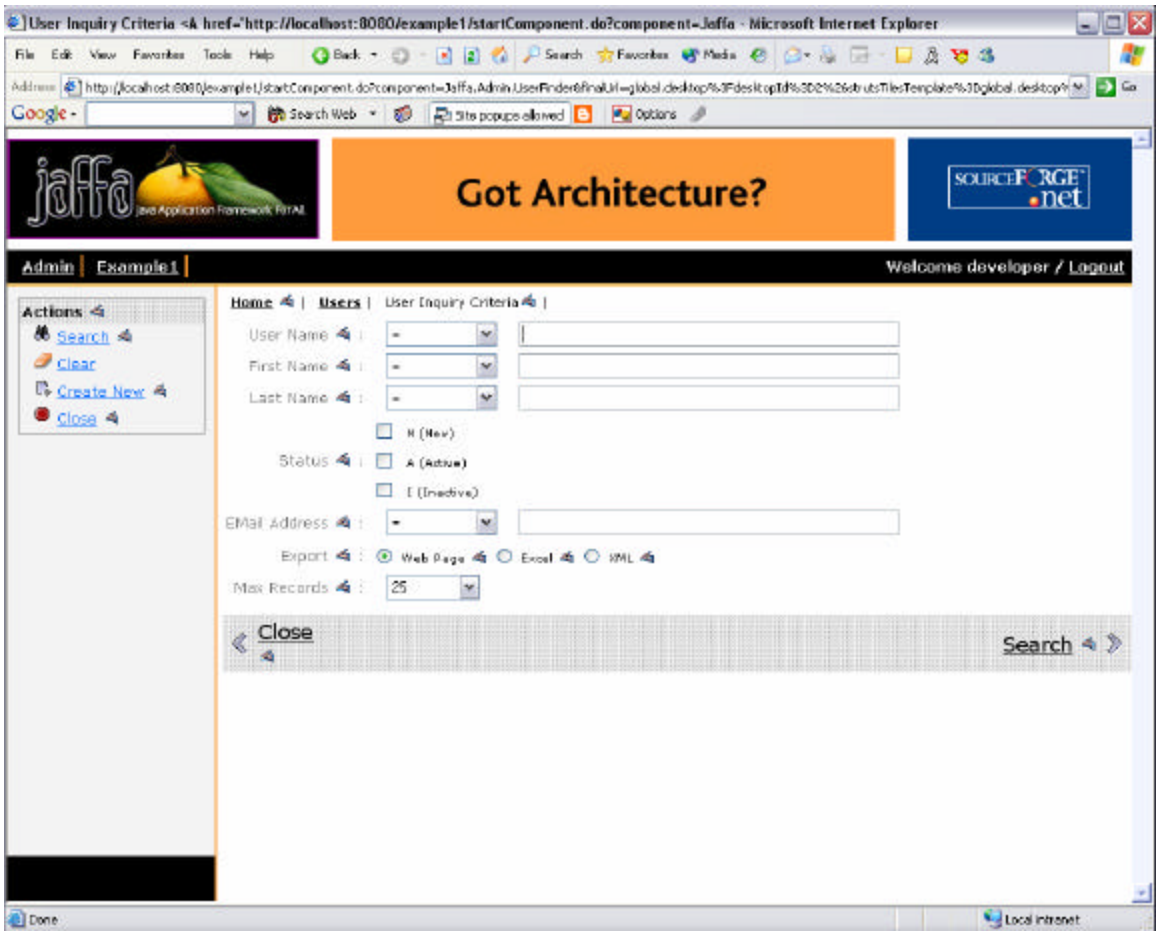
The UserFinder component can be invoked by the URL

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.UserFinder
```

When running the Jaffa baseline application, it can be accessed from

```
Admin -> Users -> User Inquiry
```

Enter a search criteria and hit ‘Search’.



Do not enter any criteria if you want to view all user account.
A list of user accounts will be displayed.
Hit the 'Update' action to update an user account. Make the required changes and hit 'Save'.
Note: The current password will not be displayed in the update mode. Enter a password only if you want to change the original value.
Hit 'Delete' to delete a user account.

3.9.2.3 Updating my account

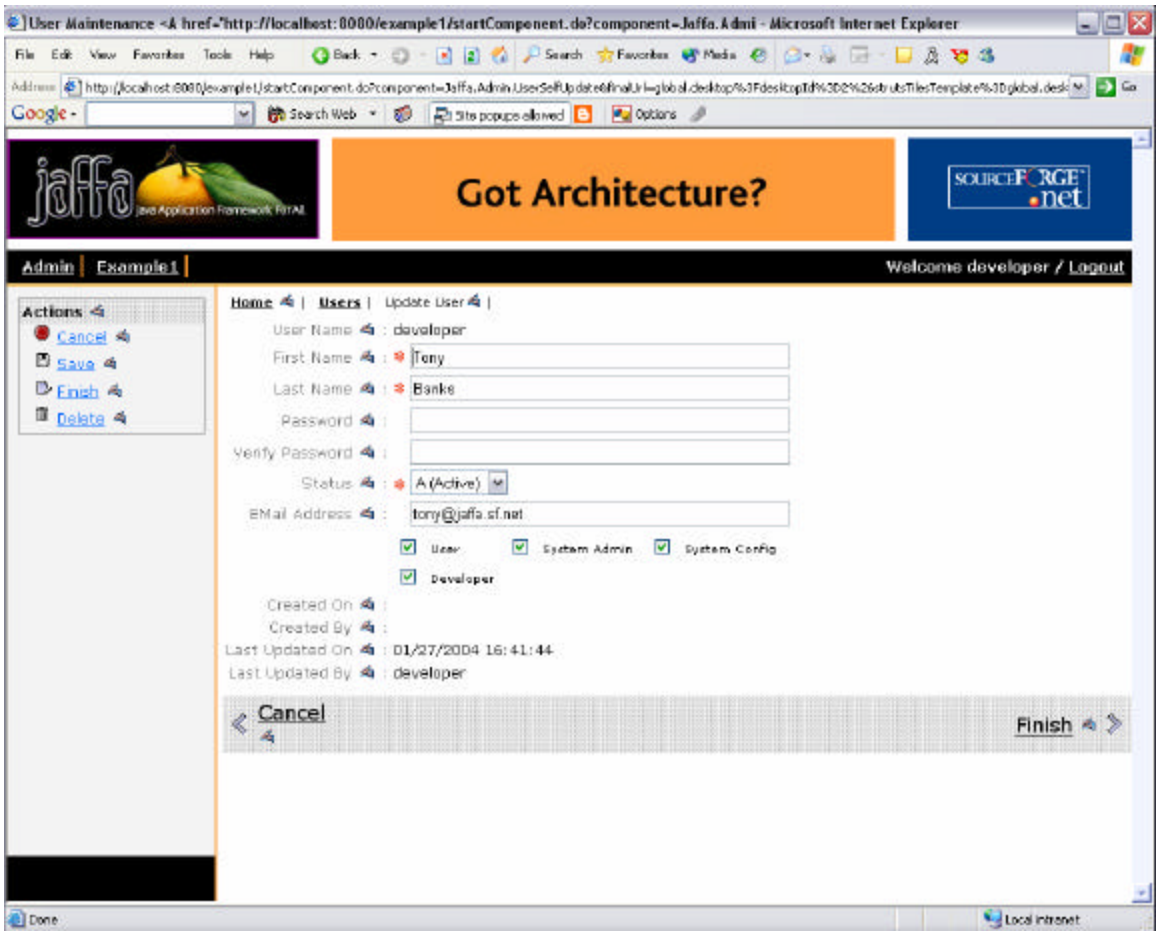
The UserSelfUpdate component can be invoked by the URL

```
http://localhost:8080/MyApp/startComponent.do?component=Jaffa.Admin.UserSelfUpdate
```

When running the Jaffa baseline application, it can be accessed from

```
Admin -> Users -> Modify my account
```

This will bring up a screen in which the user can modify his/her password and email.



Note: If the user has access to the UserMaintenance screen, then he/she can modify all the attributes of his/her user account.

3.9.3 Setup and Configuration

Required business-function(s) for a component:

- `Jaffa.Admin.UserMaintenance: Jaffa.Admin.User.Maintenance`
- `Jaffa.Admin.UserFinder: Jaffa.Admin.User.Inquiry, Jaffa.Admin.User.Details`
- `Jaffa.Admin.UserSelfUpdate: Jaffa.Admin.User.SelfUpdate`
- `Jaffa.Admin.UserViewer: Jaffa.Admin.User.Details`

The User Account is backed by 2 database tables

- USERS: This stores all the attributes for the user account
- USER_ROLE: A user will have one or more records in this table, depending on the assigned roles

The `'jdbcengine/User.xml'` mapping file maps the User domain object to the USERS table in the database

```
<classmap>
    <class name="org.jaffa.applications.jaffa.modules.admin.domain.User">
        <map-to table="USERS" />

        <field name="UserName" type="java.lang.String" primary-key="true">
            <sql name="USER_NAME" type="STRING" dirty="ignore"/>
        </field>
        . . .
        . . .
        . . .
    </class>
</classmap>
```

The '`jdbcengine/UserRole.xml`' mapping file maps the UserRole domain object to the USER_ROLE table in the database

```
<classmap>
    <class name="org.jaffa.applications.jaffa.modules.admin.domain.UserRole">
        <map-to table="USER_ROLE" />

        <field name="UserName" type="java.lang.String" primary-key="true">
            <sql name="USER_NAME" type="STRING" dirty="ignore"/>
        </field>

        <field name="RoleName" type="java.lang.String" primary-key="true">
            <sql name="ROLE_NAME" type="STRING" dirty="ignore"/>
        </field>

    </class>
</classmap>
```

Change the value for the <map-to> element to specify a different table, if required.

Check the scripts in 'source/sql/create-table-scripts' folder of the JaffaComponents project for creating the tables.
Check the scripts in 'source/sql/install' folder of the JaffaComponents project for loading some sample user accounts.

4 Web Services

4.1 Overview

Simple web service integration is the part of the next phase of the Jaffa Components project. Watch this space