NetAdvantage for JSF

Table of Contents

NetAdvantage for JSF

1 Getting Started

This section provides the answers to your key questions on getting started using the NetAdvantage for JSF components. It guides you through fundamental tasks such as installing the product and samples, importing the components into an IDE, etc.

- **Welcome to NetAdvantage for JSF (Section 1.1)** -- Introduces you to the NetAdvantage for JSF product.
- **Supported Software (Section 1.2)** -- Lists the software that the NetAdvantage for JSF product supports.
- **IDE Support (Section 1.3)** -- Contains links to procedures on how to set up the NetAdvantage for JSF components in a specific IDE.
- **Portal Support (Section 1.4)** -- Contains links to procedures on how to set up the NetAdvantage for JSF components in a specific portal.
- **NetAdvantage for JSF Components, JSF 1.1 and JSF 1.2 (Section 1.5)** -- Contains information relating to the JSF 1.1 and JSF 1.2 frameworks and how the NetAdvantage for JSF components work in them.
- **Deploying Your Application (Section 1.6)** -- Provides information on how to deploy your Web application.
- **Installing and Deploying the Samples (Section 1.7)** -- Provides information on how to install the demos and deploy them to your Web application server.
- **Managing the License (Section 1.8)** -- Provides information on where the license file must reside in order to use the NetAdvantage for JSF product.
- **Localization (Section 1.9)** – Provides information on the workaround to a Sun Microsystems™ known issue regarding the <f: loadBundle> component, which is used for localization.
- **Getting Support (Section 1.10)** -- Provides information on the support services provided by Infragistics.

1.1 Welcome to NetAdvantage for JSF

The NetAdvantage for JSF product is a comprehensive set of AJAX-enabled JavaServer™ Faces (JSF) components for building commercial-class user interfaces for J2EE™ applications.

NetAdvantage for JSF has every major UI control including a hierarchical grid, charting, calendar, menu, tree, tab, listbar, dialog window, drop down and input controls. All components are AJAX-enabled, with built in styles and fully customizable look and feel.

IDE design times support include:

- Eclipse 3.0 and later
- IBM® Rational® Application Developer for WebSphere Software® 7.0

The NetAdvantage for JSF product runs on all the major application servers including:

- BEA WebLogic™
- IBM WebSphere®
- JBoss®
- Apache® Tomcat
- Sun® Java System Application Server

1.2 Supported Software

The NetAdvantage for JSF product is supported for the following software:

- Java™
    - JDK™ 1.4.x and up
- JavaServer™ Faces Reference Implementation (RI)
    - Sun® RI 1.0 and later
    - Apache™ MyFaces 1.1.5, 1.2.3
    - JavaServer Faces 1.1, 1.2
- Portal Servers
    - JBoss® 2.4.*, 2.6.*
    - IBM® WebSphere® 6.0.1 (Production), 6.1 with AJAX
    - BEA® WebLogic™ 10.0 (Production) with AJAX
- Application Servers
    - Apache™ Tomcat 5.5, 6.0
    - JBoss Application Server 4.2.2 GA, 5.0.0 BETA 4
    - Sun Application Server 9.1
    - IBM WebSphere Application Server 6.1
    - BEA WebLogic Application Server 10.0
- Integrated development environment (IDE)
    - Eclipse™ 3.0 and later
    - IBM Rational® Application Developer for WebSphere Software® 7.0
- Web browsers (for AJAX support)
    - Microsoft® Internet Explorer 6 and 7
    - Mozilla® Firefox® 1.5 and 2.0
- Operating Systems
    - Suse® 10
    - Microsoft Windows® XP, Vista™, Server 2003

1.3 IDE Support

NetAdvantage for JSF components are fully integrated into the available JSP page designer tools palette for IBM® Rational® Application Developer for WebSphere® 7.0.

Click the following link to learn how to install the plug-in for the IDE, as well as set up an application in the IDE:

- **IBM RAD (Section 1.3.1)**

Click the following link to learn how to set up the NetAdvantage for JSF components in Eclipse:

- **Eclipse (Section 1.3.2)**

1.3.1 IBM RAD

NetAdvantage for JSF components are integrated with IBM® Rational® Application Developer for WebSphere® 7.0, allowing you to take full advantage of the design surface of RAD. You can drag the NetAdvantage for JSF components from the palette to your JSP page.

Click the following links to learn how to install the NetAdvantage for JSF plug-in for RAD and how to set up an application using our components in RAD:

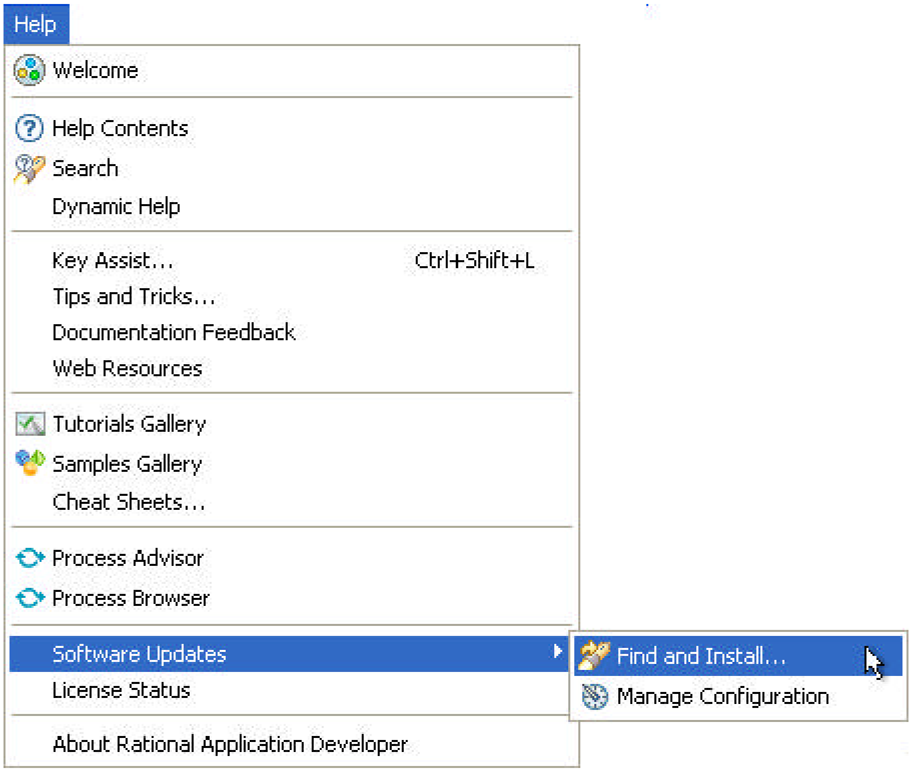
- **Installing the NetAdvantage for JSF Plug-In for IBM RAD (Section 1.3.1.1)**
- **Setting Up Your Application in IBM RAD (Section 1.3.1.2)**
- **Upgrading the NetAdvantage for JSF Plug-In for IBM RAD (Section 1.3.1.3)**

1.3.1.1 Installing the NetAdvantage for JSF Plug-In for IBM RAD

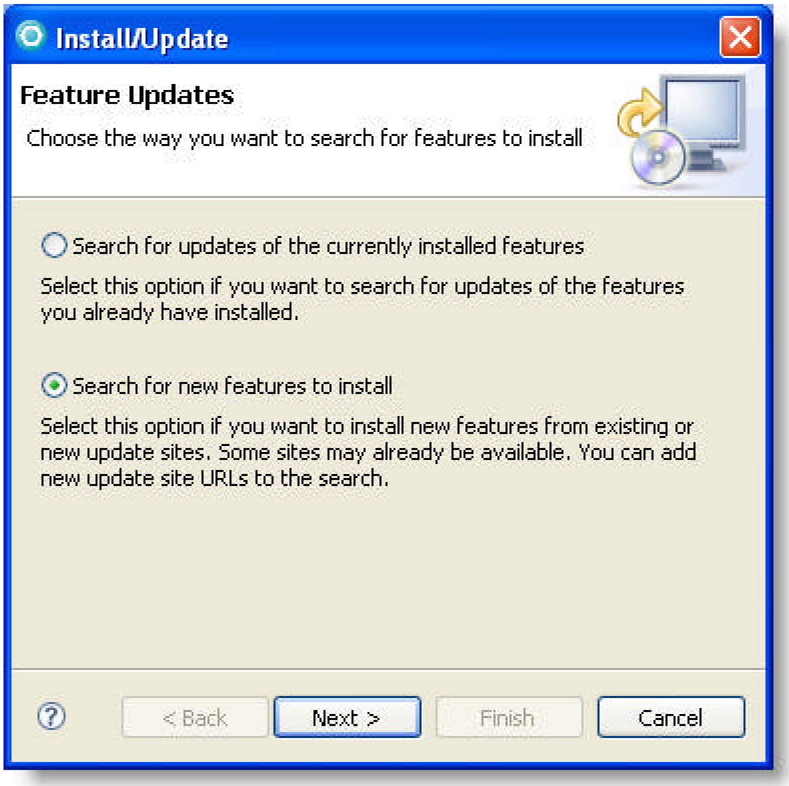
Starting in the 2007 Volume 2 release, the NetAdvantage for JSF components are fully integrated into the available JSP page designer tools palette for IBM® Rational® Application Developer for WebSphere 7.0. This integration with the IBM RAD IDE lets you create an application quickly and easily using the design view.

To install the NetAdvantage for JSF plug-in into IBM RAD:

1. Open the IBM RAD IDE.
2. On the Help menu, point to Software Updates and click Find and Install...



3. In the Install/Update wizard, select the "Search for new features to install" option, then click Next.



Feature Updates

Choose the way you want to search for features to install

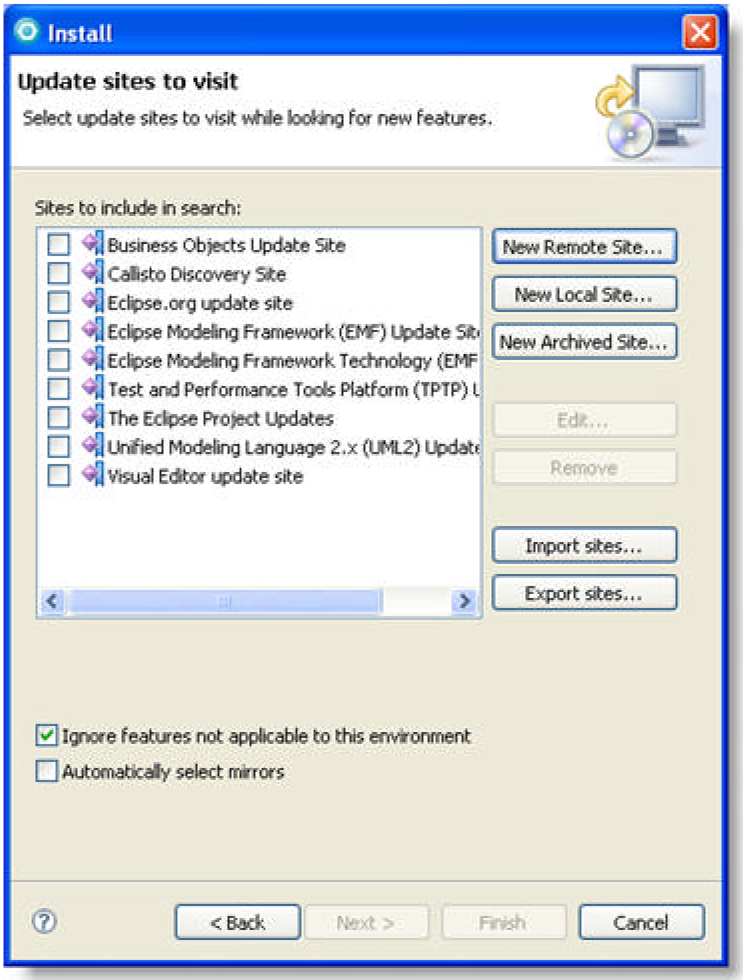
○ Search for updates of the currently installed features

Select this option if you want to search for updates of the features you already have installed.

● Search for new features to install

Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.

4. In the wizard, click New Local Site...



Install

Update sites to visit

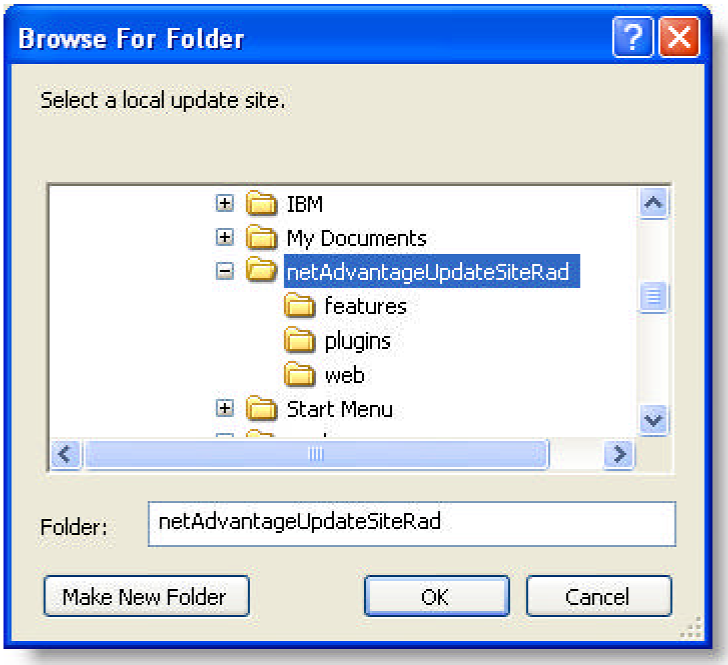
Select update sites to visit while looking for new features.

Sites to include in search:

☐ Business Objects Update Site
☐ Callisto Discovery Site
☐ Eclipse.org update site
☐ Eclipse Modeling Framework (EMF) Update Sit
☐ Eclipse Modeling Framework Technology (EMF
☐ Test and Performance Tools Platform (TPTP) U
☐ The Eclipse Project Updates
☐ Unified Modeling Language 2.x (UML2) Update
☐ Visual Editor update site

☑ Ignore features not applicable to this environment
☐ Automatically select mirrors

5. Browse the the netAdvantageUpdateSiteRad folder on your hard drive, select it, and click OK. This is located in the INSTALL_PATH\lib\designtime folder.

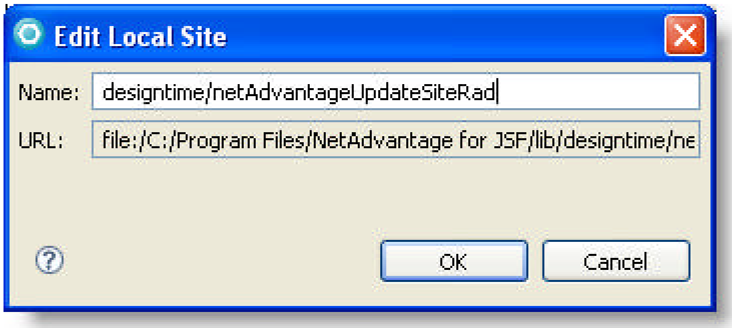


Browse For Folder

Select a local update site.

Folder: netAdvantageUpdateSiteRad

6. In the Edit Local Site dialog box that appears, click OK.

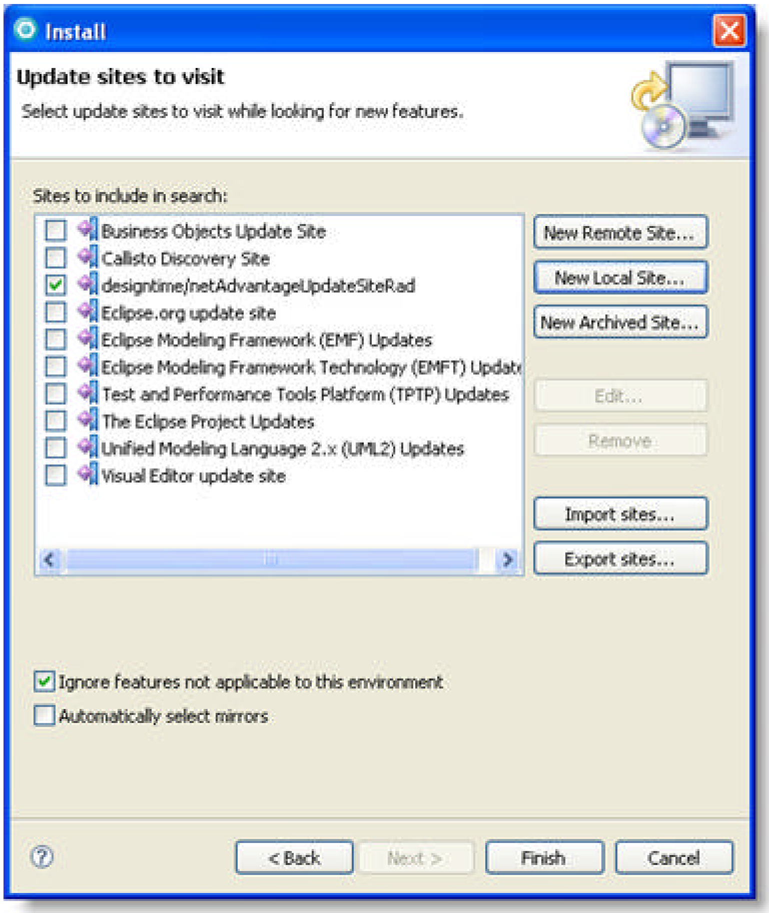


Edit Local Site

Name: designtime/netAdvantageUpdateSiteRad

URL: file:/C:/Program Files/NetAdvantage for JSF/lib/designtime/ne

7. In the Update Sites to Visit page, select the designtime/netAdvantageUpdateSiteRad check box, and click Finish.



Install

Update sites to visit

Select update sites to visit while looking for new features.

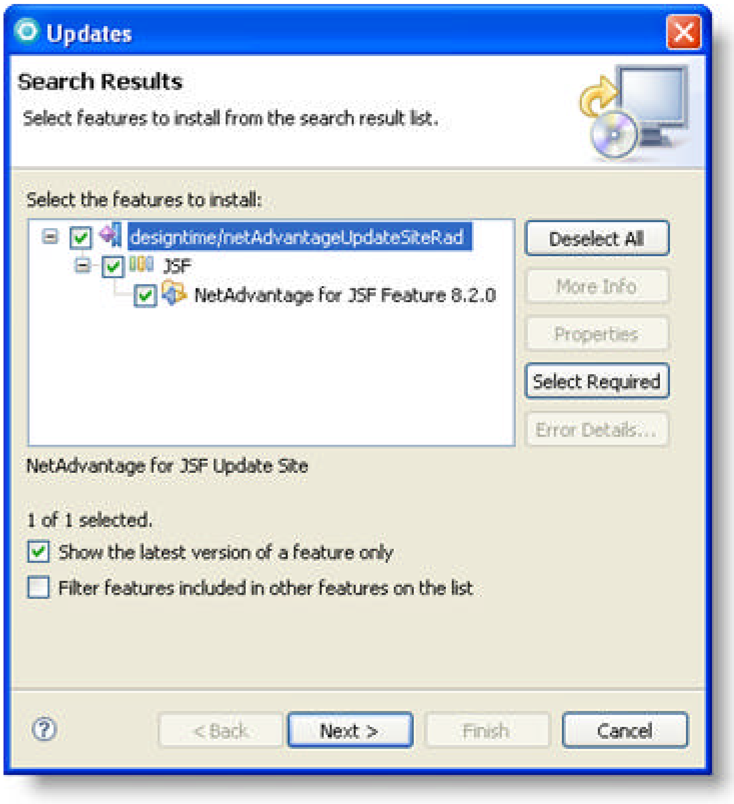
Sites to include in search:

- □ Business Objects Update Site
- □ Callisto Discovery Site
- ☑ designtime/netAdvantageUpdateSiteRad
- □ Eclipse.org update site
- □ Eclipse Modeling Framework (EMF) Updates
- □ Eclipse Modeling Framework Technology (EMFT) Updat
- □ Test and Performance Tools Platform (TPTP) Updates
- □ The Eclipse Project Updates
- □ Unified Modeling Language 2.x (UML2) Updates
- □ Visual Editor update site

☑ Ignore features not applicable to this environment

□ Automatically select mirrors

8. In the Search Results page of the wizard, select the designTime/netAdvantageUpdateSiteRad check box. This should automatically select all the tree.



Updates

Search Results

Select features to install from the search result list.

Select the features to install:

- ☑ designtime/netAdvantageUpdateSiteRad
  - ☑ JSF
    - ☑ NetAdvantage for JSF Feature 8.2.0

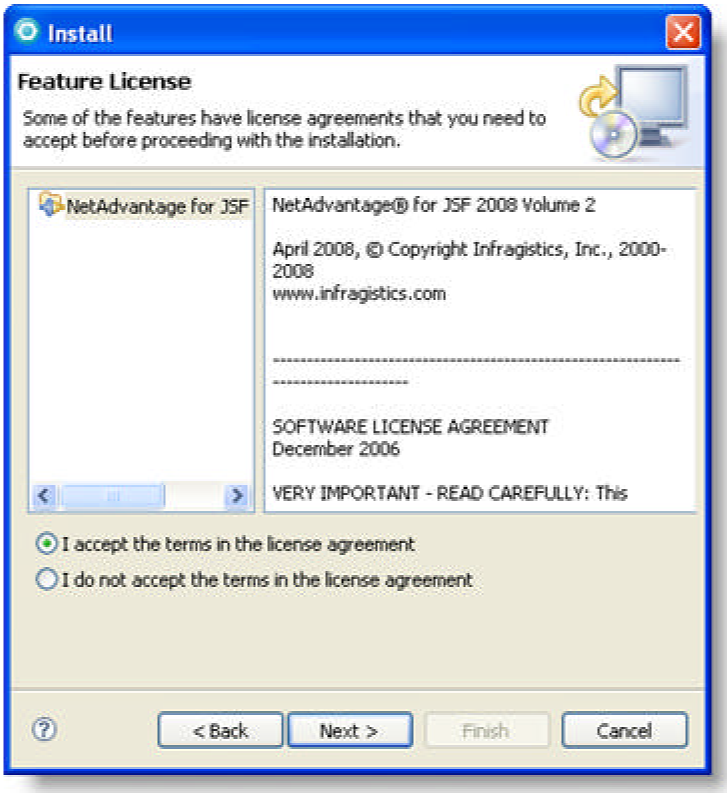
NetAdvantage for JSF Update Site

1 of 1 selected.

☑ Show the latest version of a feature only

□ Filter features included in other features on the list

9. Click Next.
10. In the Feature License page of the wizard, select the "I accept the terms in the license agreement", and click Next.



Feature License

Some of the features have license agreements that you need to accept before proceeding with the installation.

NetAdvantage® for JSF 2008 Volume 2

April 2008, © Copyright Infragistics, Inc., 2000-2008
www.infragistics.com

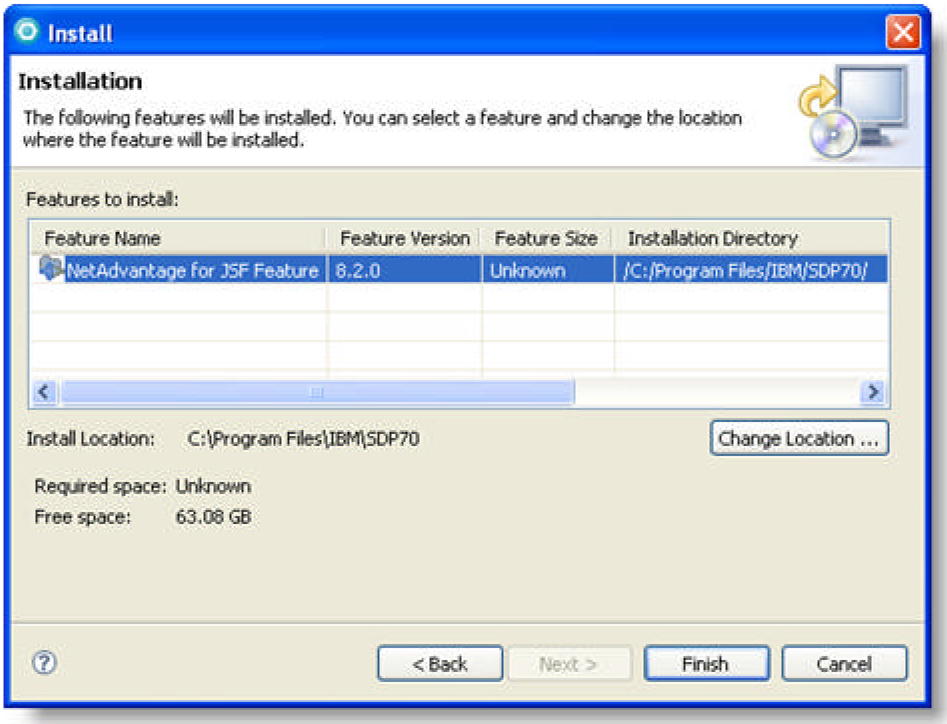
SOFTWARE LICENSE AGREEMENT
December 2006

VERY IMPORTANT - READ CAREFULLY: This

● I accept the terms in the license agreement

○ I do not accept the terms in the license agreement

11. In the Installation page of the wizard, NetAdvantage for JSF feature should appear in the Features to Install list.



Installation

The following features will be installed. You can select a feature and change the location where the feature will be installed.

Features to install:

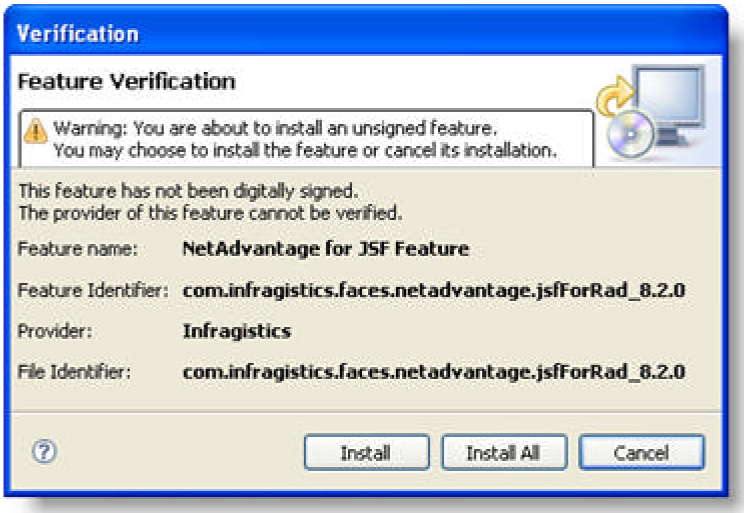
Feature Name	Feature Version	Feature Size	Installation Directory
NetAdvantage for JSF Feature	8.2.0	Unknown	/C:/Program Files/IBM/SDP70/

Install Location: C:\Program Files\IBM\SDP70

Required space: Unknown
Free space: 63.08 GB

12. Click Finish.

13. In the Verification dialog box, click Install All.



Verification

Feature Verification

Warning: You are about to install an unsigned feature.
You may choose to install the feature or cancel its installation.

This feature has not been digitally signed.
The provider of this feature cannot be verified.

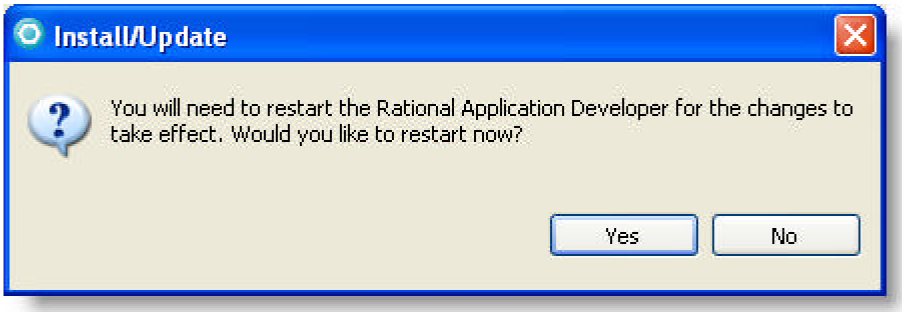
Feature name: NetAdvantage for JSF Feature

Feature Identifier: com.infragistics.faces.netadvantage.jsfForRad_8.2.0

Provider: Infragistics

File Identifier: com.infragistics.faces.netadvantage.jsfForRad_8.2.0

14. When the installation is finished, you will be prompted to restart RAD. Click Yes.



Install/Update

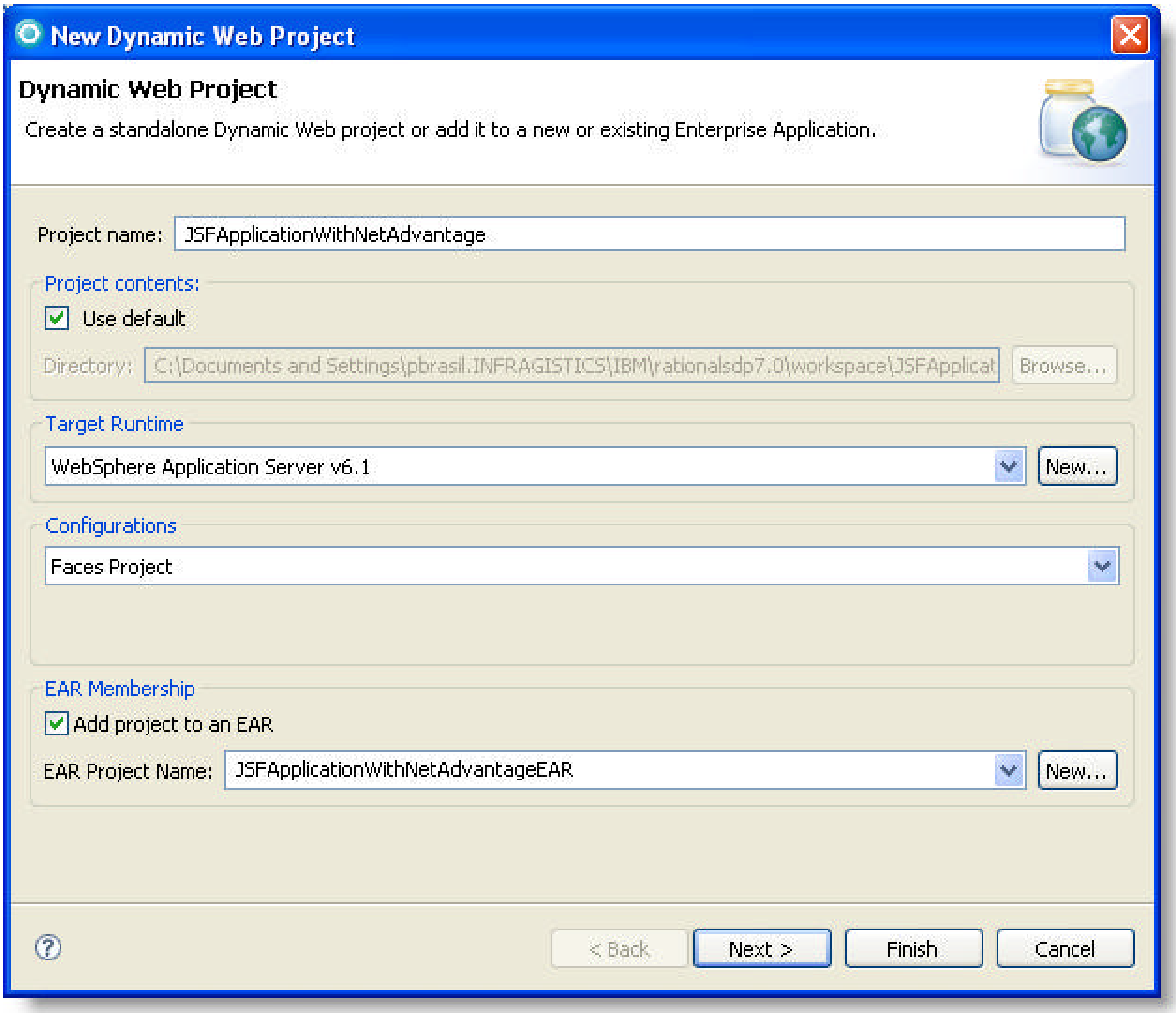
You will need to restart the Rational Application Developer for the changes to take effect. Would you like to restart now?

1.3.1.2 Setting Up Your Application in IBM RAD

This topic guides you through the process of setting up an application using the NetAdvantage for JSF components in IBM® Rational® Application Developer for WebSphere 7.0.

To set up an application using NetAdvantage for JSF components in RAD:

1. On the File menu, point to New, and click Dynamic Web Project to create a new project. The New Dynamic Web Project wizard appears. In the first page of the New Dynamic Web Project wizard, enter the following information, and click Next:
   - Project name - Enter a name for the project.
   - Project contents - If you want the new project to be created in a workspace other than the current one, deselect the "Use default" check box, and click Browse... to navigate to the new workspace. By default, the "Use default" check box is selected.
   - Target Runtime - Select a target server for your application. If your server is not listed in the drop-down list, click New... to navigate to a new server.
   - Configurations - From the drop-down list, select Faces Project for myfaces support.
   - EAR Membership - If you want to add Web components to an enterprise application (EAR file), select the "Add project to an EAR" check box. To select a different EAR project, click New... to browse to the project.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: JSFApplicationWithNetAdvantage

Project contents:

☑ Use default

Directory: C:\Documents and Settings\pbrasil.INFRAGISTICS\IBM\rationalsdp7.0\workspace\JSFApplicat

Target Runtime

WebSphere Application Server v6.1

Configurations

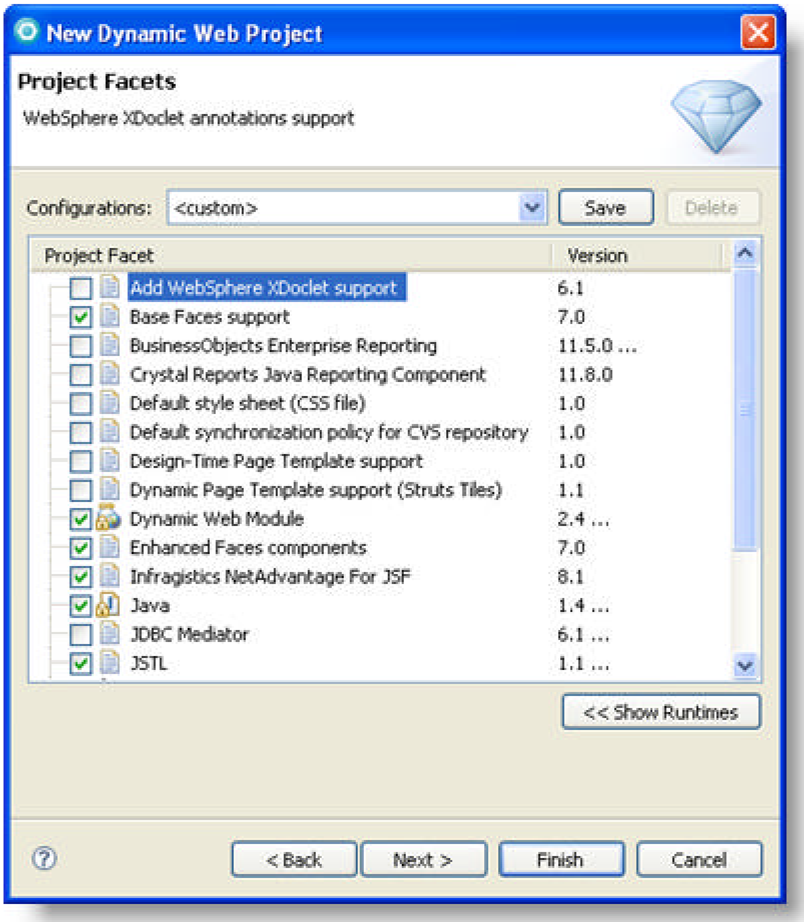
Faces Project

EAR Membership

☑ Add project to an EAR

EAR Project Name: JSFApplicationWithNetAdvantageEAR

2. In the Project Facets page of the wizard, select the Infragistics NetAdvantage For JSF check box, and click Next.



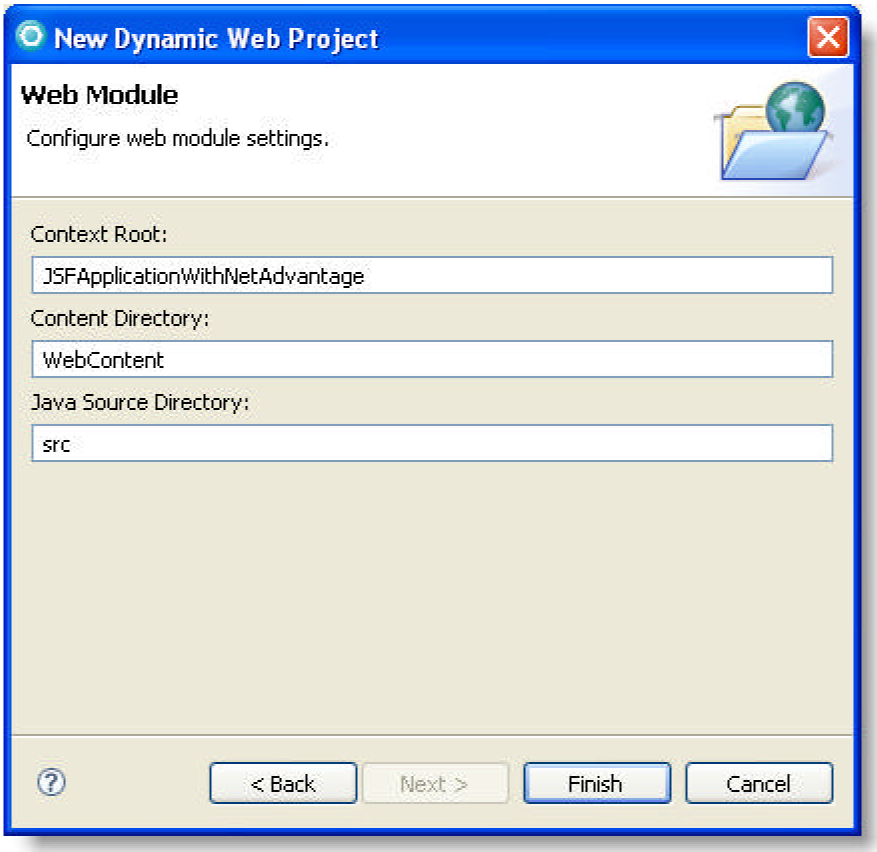
Project Facets

WebSphere XDoclet annotations support

Configurations: <custom>

Project Facet	Version
☐ Add WebSphere XDoclet support	6.1
☑ Base Faces support	7.0
☐ BusinessObjects Enterprise Reporting	11.5.0 ...
☐ Crystal Reports Java Reporting Component	11.8.0
☐ Default style sheet (CSS file)	1.0
☐ Default synchronization policy for CVS repository	1.0
☐ Design-Time Page Template support	1.0
☐ Dynamic Page Template support (Struts Tiles)	1.1
☑ Dynamic Web Module	2.4 ...
☑ Enhanced Faces components	7.0
☑ Infragistics NetAdvantage For JSF	8.1
☑ Java	1.4 ...
☐ JDBC Mediator	6.1 ...
☑ JSTL	1.1 ...

3. In the Web Module page of the wizard, enter the following information, then click Finish:
   - Context Root - This specifies the Web application root, the top-level directory of your application when it is deployed to a Web server. The default value is the name of your Web project.
   - Content Directory - This specifies the name of the directory where the common Web resources are located.
   - Java Source Directory - This specifies the name of the directory where the Java source files are located.



Web Module

Configure web module settings.

Context Root:

JSFApplicationWithNetAdvantage

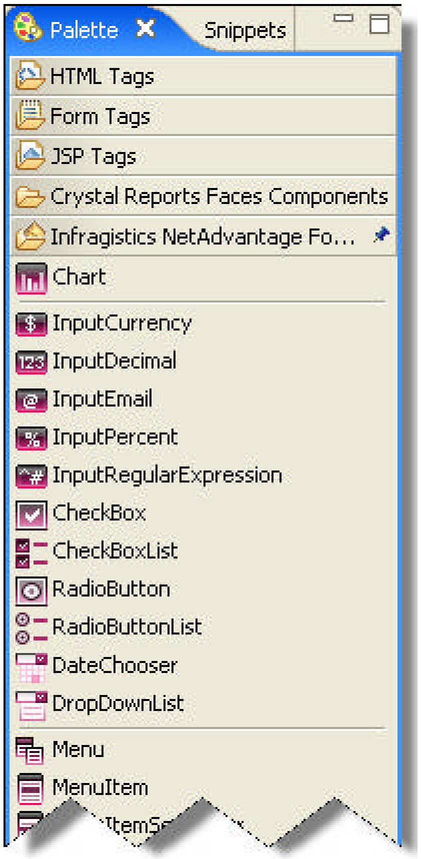
Content Directory:

WebContent

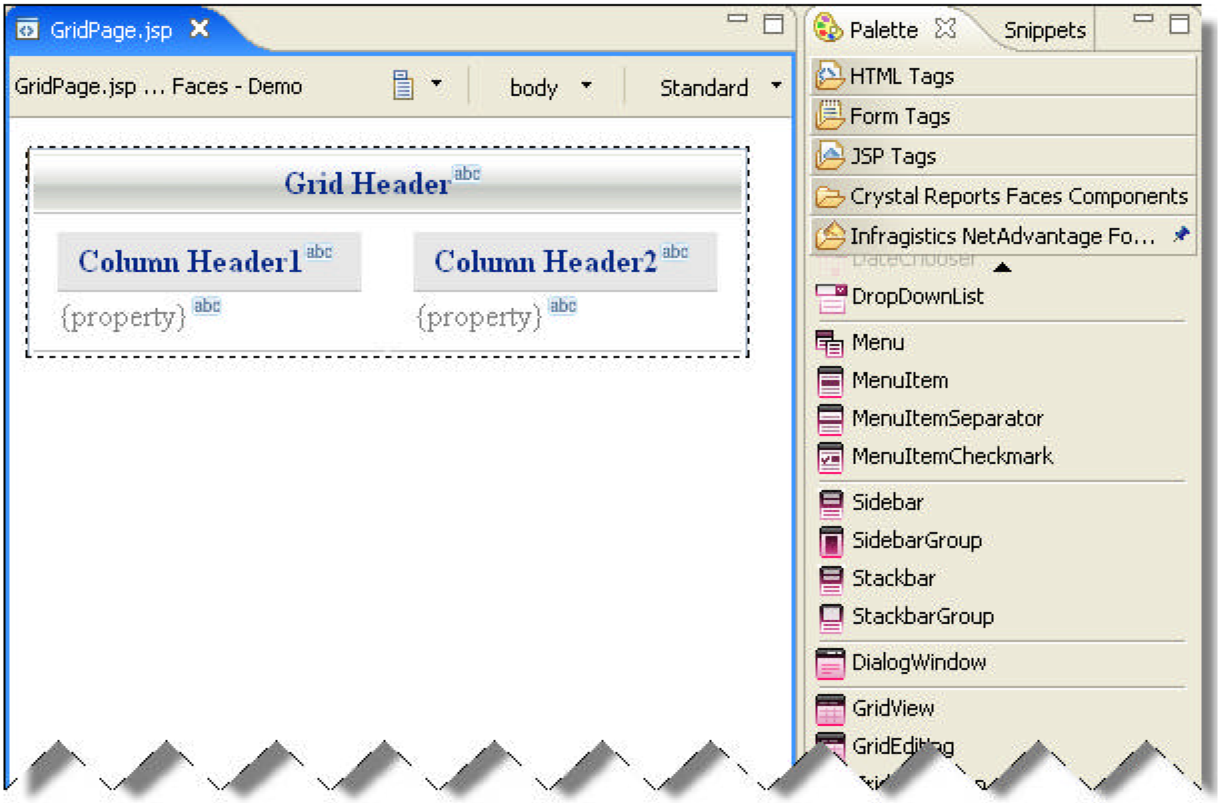
Java Source Directory:

src

4. Create a new JSP page in your application. You should see a new tab in the Palette that displays all the NetAdvantage for JSF components. You can then drag the components onto your JSP page.



5. Create a new JSP page in your application. You should see a new tab in the Palette that displays all the NetAdvantage for JSF components. You can then drag the components onto your JSP page.

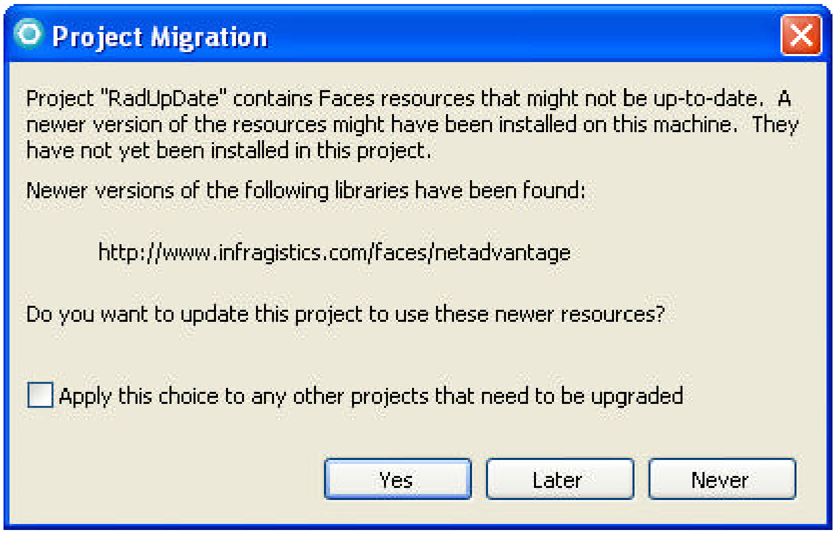

GridPage.jsp
GridPage.jsp ... Faces - Demo
body
Standard
Grid Header abc
Column Header1 abc
Column Header2 abc
{property} abc
{property} abc
Palette
Snippets
HTML Tags
Form Tags
JSP Tags
Crystal Reports Faces Components
Infragistics NetAdvantage Fo...
DateChooser
DropDownList
Menu
MenuItem
MenuItemSeparator
MenuItemCheckmark
Sidebar
SidebarGroup
Stackbar
StackbarGroup
DialogWindow
GridView
GridEditing

1.3.1.3 Upgrading the NetAdvantage for JSF Plug In for IBM RAD

This topic explains how to upgrade the NetAdvantage for JSF plug in for IBM® Rational® Application Developer for WebSphere 7.0 to a newer version of the plug-in. This ensures that all your projects created with an older version of the plug-in are still compatible with a new version of the plug-in

To upgrade the NetAdvantage for JSF plug-in into IBM RAD:

1. After installing the latest NetAdvantage for JSF plug-in for IBM RAD, open the IBM RAD IDE.
2. You will be prompted to update your projects using the newer resources. Click Yes.



Project Migration

Project "RadUpDate" contains Faces resources that might not be up-to-date. A newer version of the resources might have been installed on this machine. They have not yet been installed in this project.

Newer versions of the following libraries have been found:

http://www.infragistics.com/faces/netadvantage

Do you want to update this project to use these newer resources?

☐ Apply this choice to any other projects that need to be upgraded

3. You have successfully upgraded your NetAdvantage for JSF plug-in.

1.3.2 Eclipse

The NetAdvantage for JSF components have been designed to support the Eclipse 3.0 (and later) IDE.

To import the NetAdvantage for JSF components into Eclipse:

1. From the INSTALL_DIRECTORY/lib/runtime folder, copy the following JAR files, and paste them into your web application's WEB-INF/lib folder:
    - infragistics-3dengine.jar
    - infragistics-netadvantage.jar
    - infragistics-shared.jar
    - infragistics-webbar.jar
    - infragistics-webchart.jar
    - infragistics-webdialogwindow.jar
    - infragistics-webgrid.jar
    - infragistics-webinput.jar
    - infragistics-webmenu.jar
    - infragistics-webtab.jar
    - infragistics-webtree.jar
    - jogl.jar
    - portlet.jar
2. Copy the lib/runtime/resources folder, and paste it into the WebContent folder of your web application.
3. You are now ready to use the NetAdvantage for JSF components in Eclipse.

1.4 Portal Support

NetAdvantage for JSF components are fully supported within BEA WebLogic™ 10, IBM® WebSphere® 6.0.1 (Production), IBM WebSphere Application Server 6.1, JBoss® 2.4.* and JBoss 2.6.* portal environments.

Click the following links to learn how to configure and deploy NetAdvantage for JSF components on the following portals:

- **JBoss Portal (Section 1.4.1)**
- **WebLogic Portal (Section 1.4.2)**
- **WebSphere Portal (Section 1.4.3)**

1.4.1 JBoss Portal

Before you Begin

> **Note:** The files needed for this topic can be found at INSTALL_DIRECTORY\Portal Resources\Jboss

Ensure that:

- JBoss® is installed correctly, for more information on versions, see **Supported Software (Section 1.2)**.
- The server runs without any exceptions
- You can access http://localhost:8080/portal
- You can log in to the portal administrative interface with the administrator user name and password.
- You do not have the portlet.jar in the WEB\lib folder of your web application.
- You have not defined a context-param WAR_BUNDLES_JSF_IMPL in your web.xml file because we use the JSF implementation that JBoss provides.

What You Will Accomplish

After completing the following steps, your JSF components will be aware of the server side filter and all post backs from your JSF components will behave as normal AJAX requests.

Follow These Steps

1. Place the portlet-jboss.xml file in your web application's WEB-INF folder and rename it to portlet.xml.
2. Place the jsf-portletbridge jar into your web application's WEB-INF\lib folder.

   > **Note:** Ensure that the JSF implementation jars and the JSF API jars do not exist in the WEB-INF\lib folder as JBoss already has these jars available and are loaded at runtime.

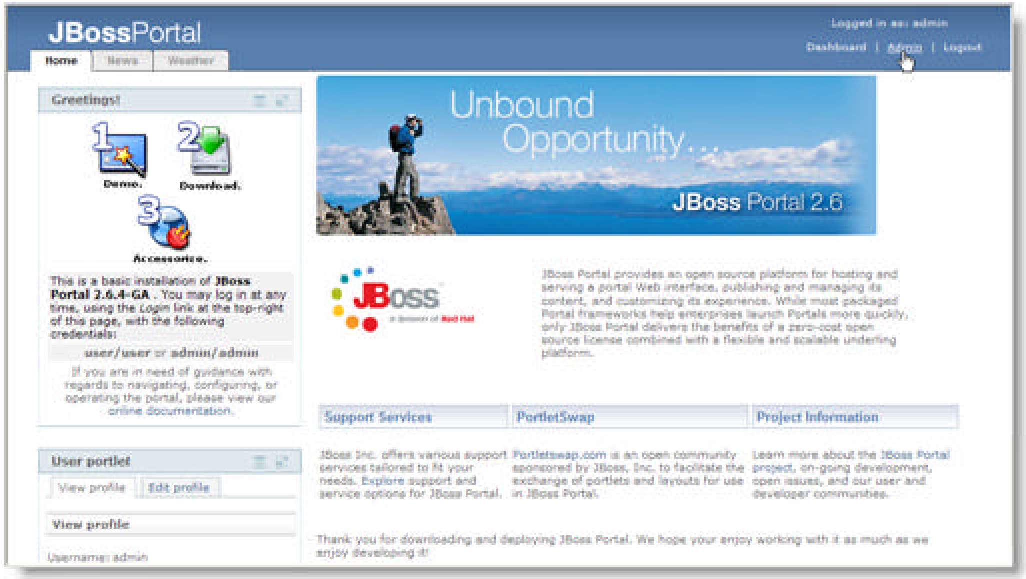
3. Place igPortalFilter.jar into the JBOSS_HOME\server\default\lib folder.
4. Open the web.xml file located at JBOSS_HOME\server\default\deploy\jboss-portal.sar\portal-server.war\WEB-INF\web.xml and add the following code:
   In XML:

```
<filter>
   <filter-name>
      IGAjaxFilter
   </filter-name>
   <filter-class>
      com.infragistics.faces.portals.JBossFilter
   </filter-class>
</filter>
<filter-mapping>
   <filter-name>
      IGAjaxFilter
   </filter-name>
   <url-pattern>
```

```
        /*
    </url-pattern>
</filter-mapping>
```

The above code snippet ensures that all portal calls are intercepted by the Infragistics servlet filter. If the call is a smart refresh request then the output will be correctly filtered to include only fragments relevant to the smart refresh event that was sent from the client side.

5. Restart JBoss.
6. Deploy your WAR file to JBOSS_HOME\server\default\deploy.
7. Login to the JBoss portal home page (http://localhost:8080/portal) as an administrator .
8. Click Admin in the top right corner.



This is a basic installation of **JBoss Portal 2.6.4-GA** . You may log in at any time, using the Login link at the top-right of this page, with the following credentials:

user/user or **admin/admin**

If you are in need of guidance with regards to navigating, configuring, or operating the portal, please view our online documentation.

JBoss Portal provides an open source platform for hosting and serving a portal Web interface, publishing and managing its content, and customizing its experience. While most packaged Portal frameworks help enterprises launch Portals more quickly, only JBoss Portal delivers the benefits of a zero-cost open source license combined with a flexible and scalable underling platform.

JBoss Inc. offers various support services tailored to fit your needs. Explore support and service options for JBoss Portal.

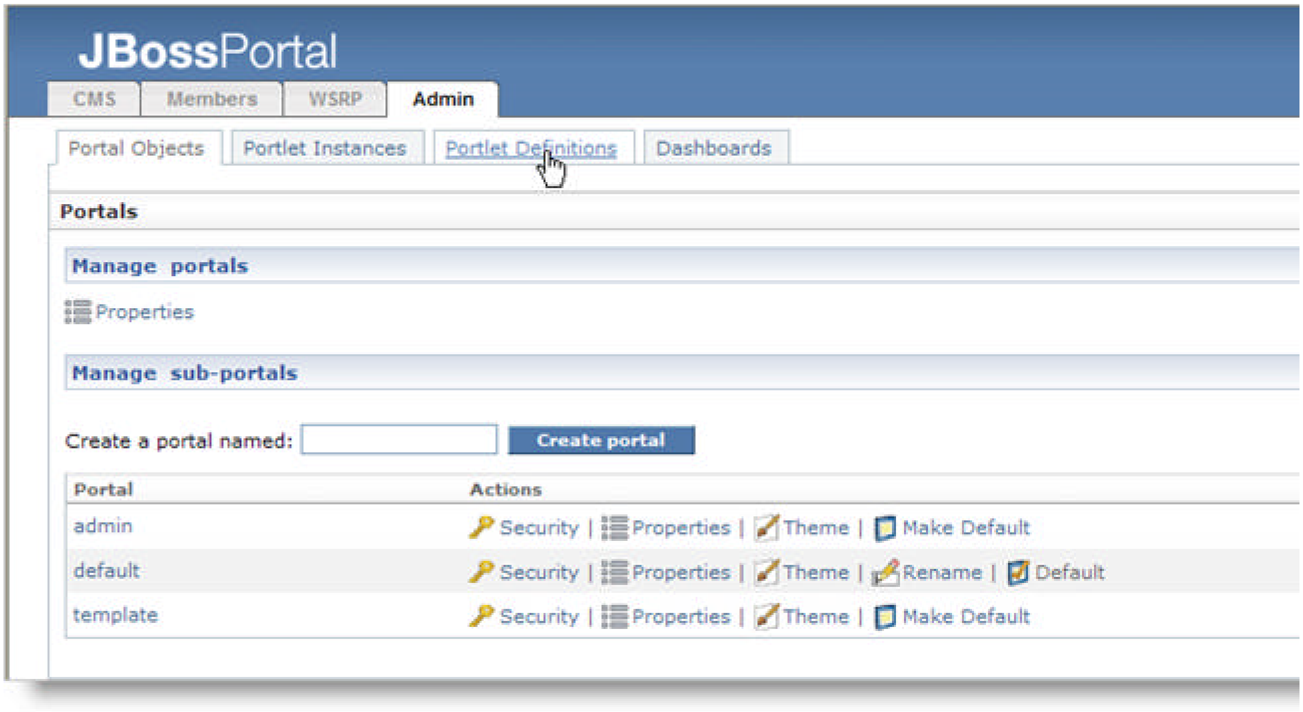
Portletswap.com is an open community sponsored by JBoss, Inc. to facilitate the exchange of portlets and layouts for use in JBoss Portal.

Learn more about the JBoss Portal project, on-going development, open issues, and our user and developer communities.

Thank you for downloading and deploying JBoss Portal. We hope your enjoy working with it as much as we enjoy developing it!

Username: admin

9. Select the Portlet Definitions tab.



Portals

Manage portals

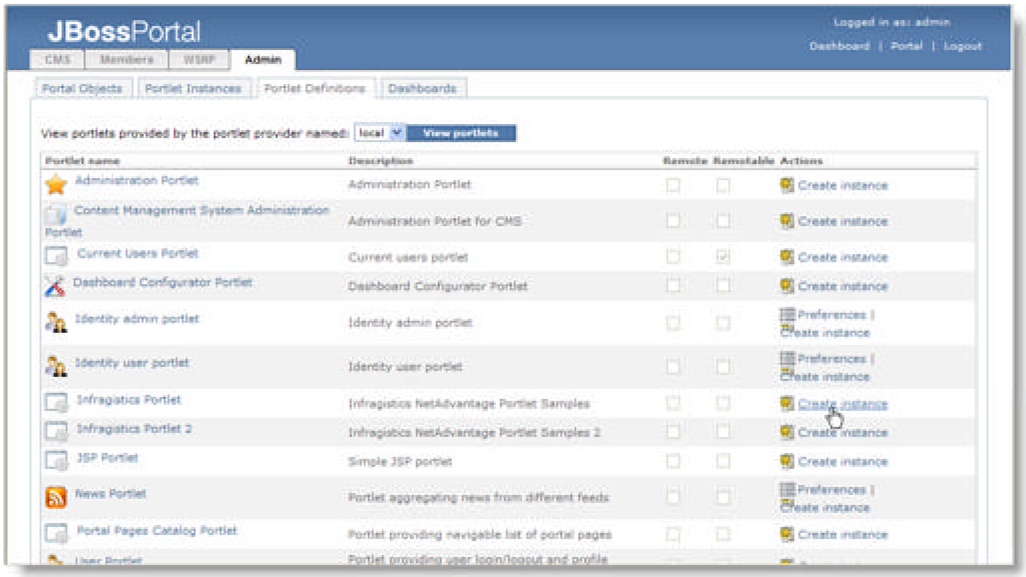
Properties

Manage sub-portals

Create a portal named: [Create portal]

Portal	Actions
admin	Security | Properties | Theme | Make Default
default	Security | Properties | Theme | Rename | Default
template	Security | Properties | Theme | Make Default

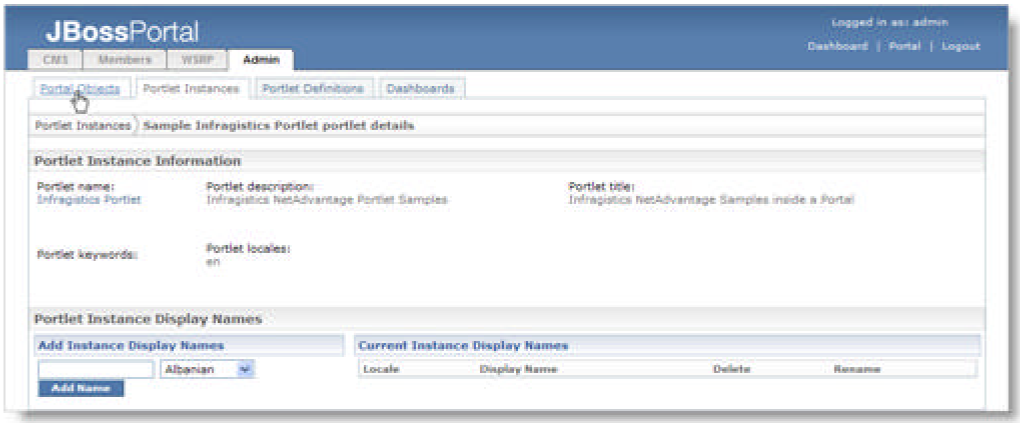
10. From the list that appears, locate the Infragistics Portlet and select Create instance.



View portlets provided by the portlet provider named: local [View portlets]

Portlet name	Description	Remote	Remotable	Actions
Administration Portlet	Administration Portlet	□	□	Create instance
Content Management System Administration Portlet	Administration Portlet for CMS	□	□	Create instance
Current Users Portlet	Current users portlet	□	☑	Create instance
Dashboard Configurator Portlet	Dashboard Configurator Portlet	□	□	Create instance
Identity admin portlet	Identity admin portlet	□	□	Preferences | Create instance
Identity user portlet	Identity user portlet	□	□	Preferences | Create instance
Infragistics Portlet	Infragistics NetAdvantage Portlet Samples	□	□	Create instance
Infragistics Portlet 2	Infragistics NetAdvantage Portlet Samples 2	□	□	Create instance
JSP Portlet	Simple JSP portlet	□	□	Create instance
News Portlet	Portlet aggregating news from different feeds	□	□	Preferences | Create instance
Portal Pages Catalog Portlet	Portlet providing navigable list of portal pages	□	□	Create instance
User Portlet	Portlet providing user login/logout and profile			

11. Enter Sample Infragistics Portlet as a name for the portlet instance. Click Create Instance.
12. The next page that appears displays the details of the Sample Infragistics Portlet.
13. Select the Portal Objects tab.



Portlet Instances 〉 Sample Infragistics Portlet portlet details

Portlet Instance Information

Portlet name: Infragistics Portlet

Portlet description: Infragistics NetAdvantage Portlet Samples

Portlet title: Infragistics NetAdvantage Samples inside a Portal

Portlet keywords:

Portlet locales: en

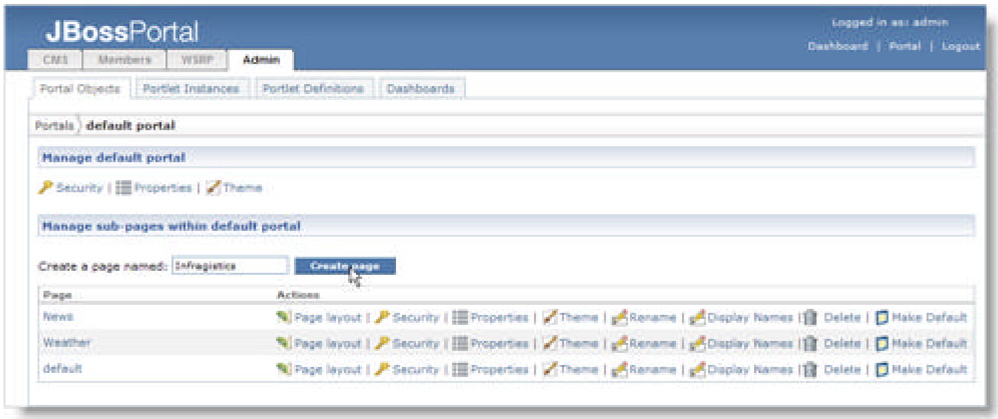
Portlet Instance Display Names

Add Instance Display Names

Current Instance Display Names

Locale	Display Name	Delete	Rename

14. From the Portal list, select default
15. Create a new sub page named Infragistics within the default portal. Click Create page.



Portals 〉 default portal

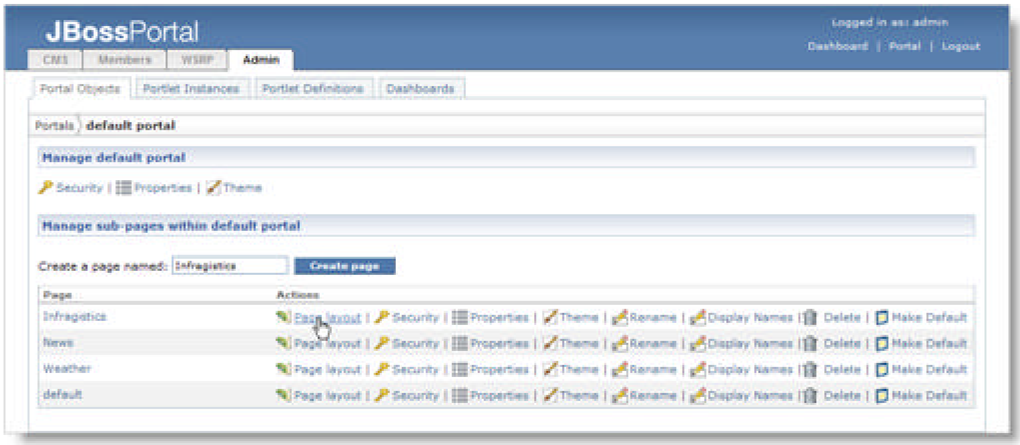
Manage default portal

Manage sub-pages within default portal

Create a page named: Infragistics

Page	Actions
News	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default
Weather	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default
default	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default

16. From the Page list, locate Infragistics and select Page Layout.



Portals 〉 default portal

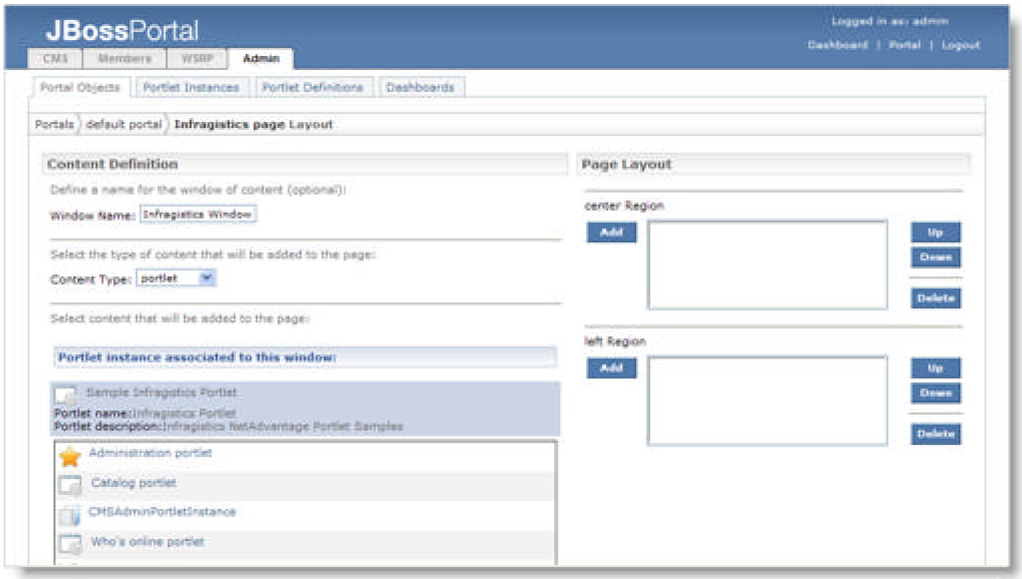
Manage default portal

Manage sub-pages within default portal

Create a page named: Infragistics

Page	Actions
Infragistics	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default
News	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default
Weather	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default
default	Page layout | Security | Properties | Theme | Rename | Display Names | Delete | Make Default

17. Under the Content Definition section, enter Infragistics Window in the Window Name text box.
18. From the portal list, select Sample Infragistics Portlet as the portlet instance associated to the Infragistics window.



Define a name for the window of content (optional):

Window Name: Infragistics Window

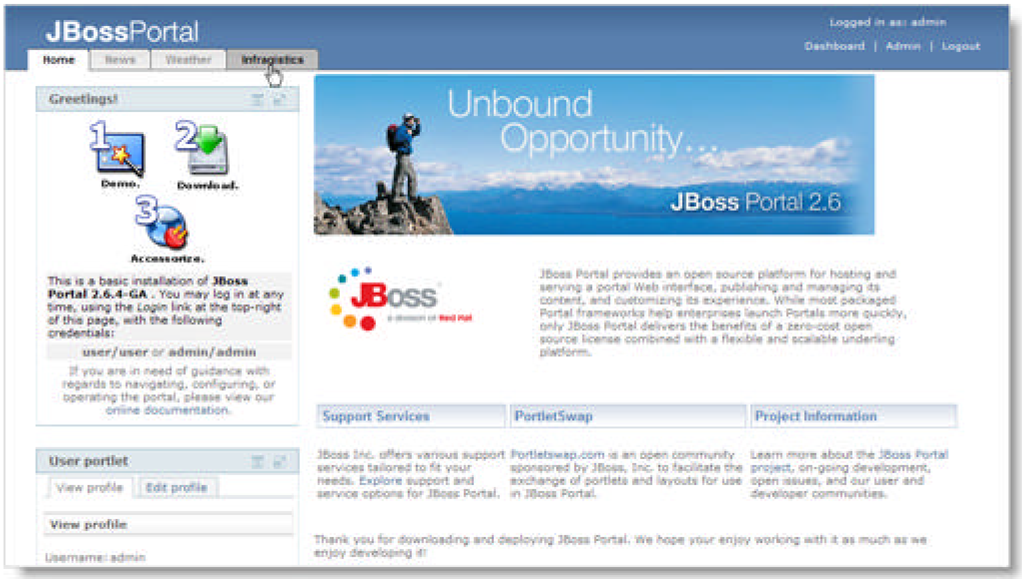
Select the type of content that will be added to the page:

Content Type: portlet

Select content that will be added to the page:

Sample Infragistics Portlet
Portlet name:Infragistics Portlet
Portlet description:Infragistics NetAdvantage Portlet Samples

19. Under the Page Layout section, click Add to add the Infragistics window to the center Region.
20. Select Portal from the top right corner.
21. Select the Infragistics tab.



This is a basic installation of **JBoss Portal 2.6.4-GA** . You may log in at any time, using the Login link at the top-right of this page, with the following credentials:

user/user or **admin/admin**

If you are in need of guidance with regards to navigating, configuring, or operating the portal, please view our online documentation.

JBoss Portal provides an open source platform for hosting and serving a portal Web interface, publishing and managing its content, and customizing its experience. While most packaged Portal frameworks help enterprises launch Portals more quickly, only JBoss Portal delivers the benefits of a zero-cost open source license combined with a flexible and scalable underling platform.

JBoss Inc. offers various support services tailored to fit your needs. Explore support and service options for JBoss Portal.

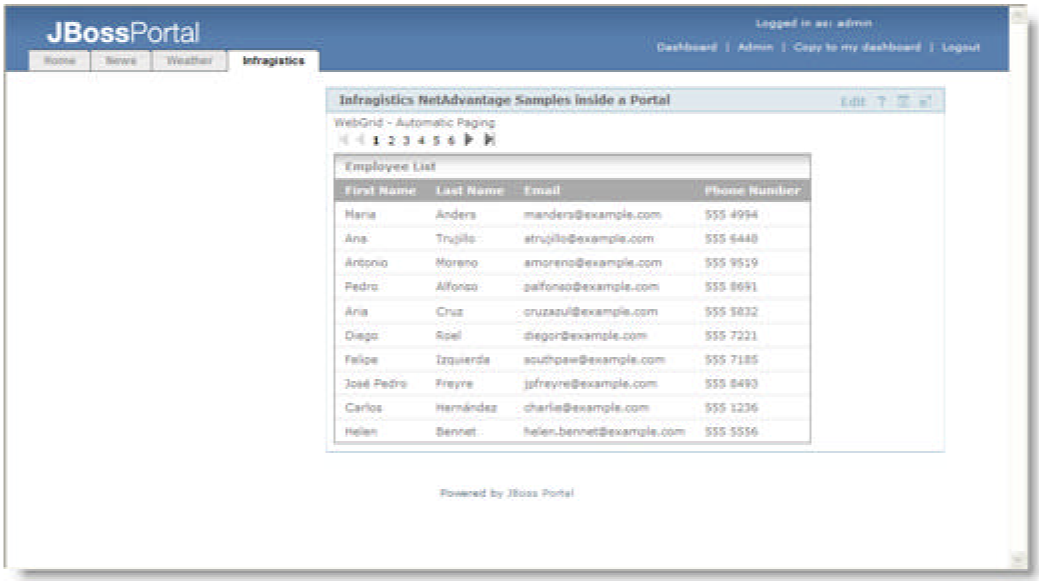
Portletswap.com is an open community sponsored by JBoss, Inc. to facilitate the exchange of portlets and layouts for use in JBoss Portal.

Learn more about the JBoss Portal project, on-going development, open issues, and our user and developer communities.

Thank you for downloading and deploying JBoss Portal. We hope your enjoy working with it as much as we enjoy developing it!

Username: admin

22. You should now see your application running inside the JBoss portal.



Infragistics NetAdvantage Samples inside a Portal

WebGrid - Automatic Paging

Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6440
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

Powered by JBoss Portal

1.4.2 WebLogic Portal

Before you Begin

Note: The files needed for this topic can be found at INSTALL_DIRECTORY\Portal Resources\WebLogic

Note: You should use the MyFaces implementation because using AJAX with Sun's RI is not supported on WebLogic Portals

Ensure that:

- BEA WebLogic™ Portal Server 10.0 is installed correctly.
- BEA Workshop for WebLogic Portal platform 10.0 is installed correctly.
- Place the portlet-weblogic.xml file in your web application's WEB-INF folder and rename it to portlet.xml.
- Place the following jars in the WEB-INF\lib folder of your web application
    - the customized myfaces-impl.jar
    - myfaces-api.jar
    - myfaces-shared-impl2.0.8.jar
- That you have the following listener configured in your web.xml file:
  In XML:

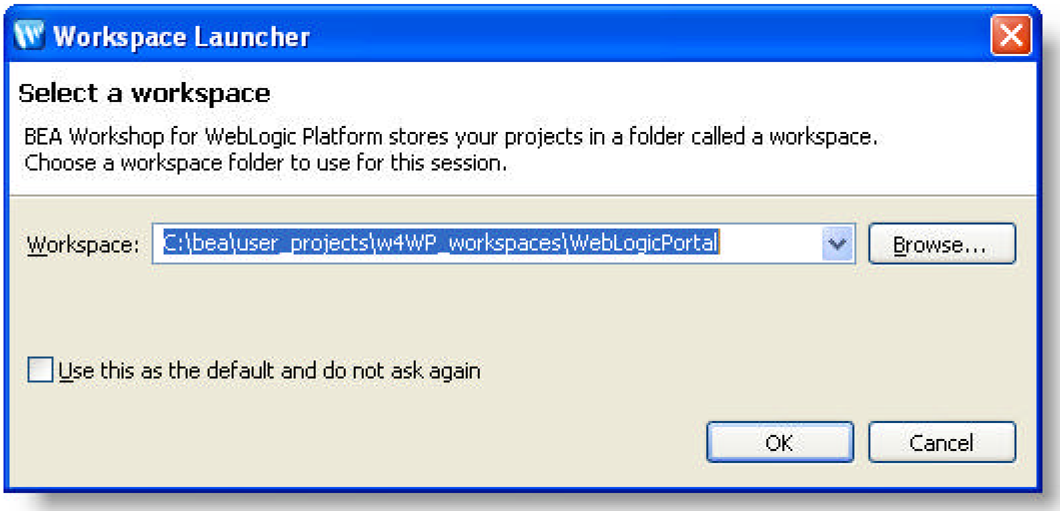
```
<listener>
   <listener-class>
      org.apache.myfaces.webapp.StartupServletContextListener
   </listener-class>
</listener>
```

What You Will Accomplish

After completing the following steps your JSF components will be aware of the server side filter and all post backs from your JSF components will behave as normal AJAX requests while running in the WebLogic portal.

Follow These Steps

1. Open BEA Workshop.
2. From the Workspace Launcher wizard, create a workspace.



Workspace Launcher

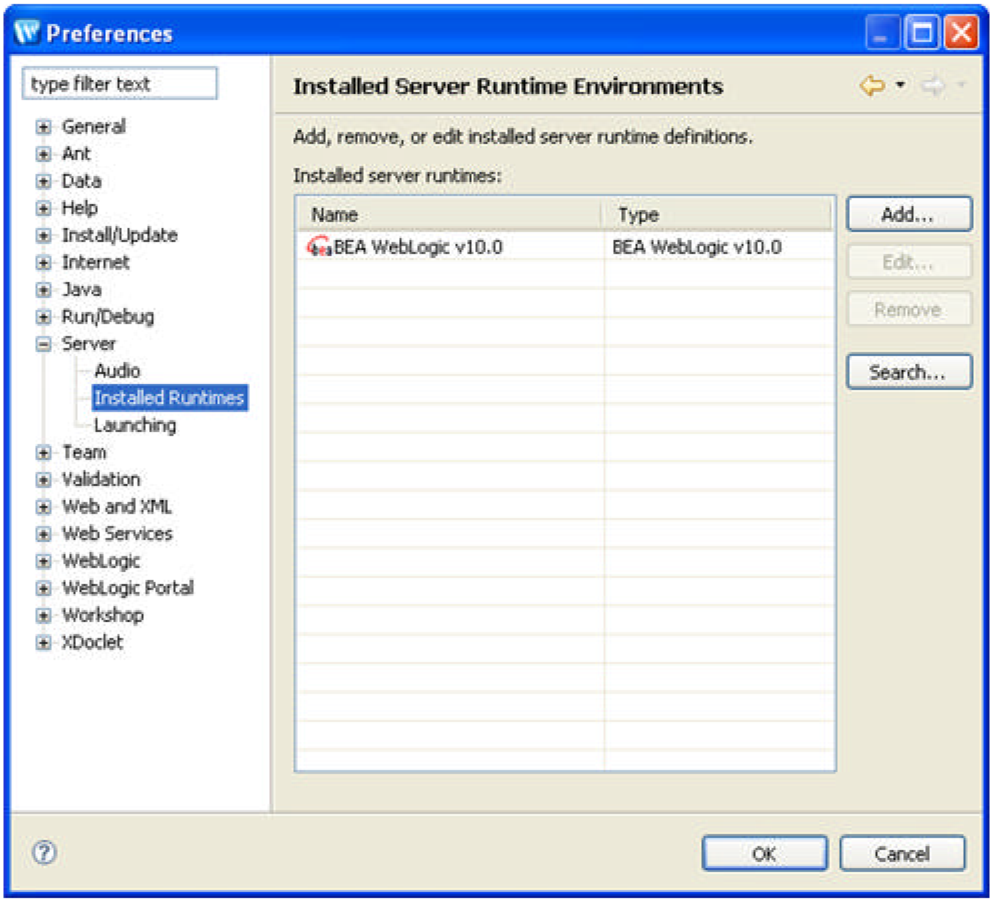
Select a workspace

BEA Workshop for WebLogic Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\bea\user_projects\w4WP_workspaces\WebLogicPortal

□ Use this as the default and do not ask again

3. On the Window menu, click to Preferences.
4. Expand the Server node, select Installed Runtimes. Ensure that the BEA WebLogic 10.0 runtime is installed.



Preferences

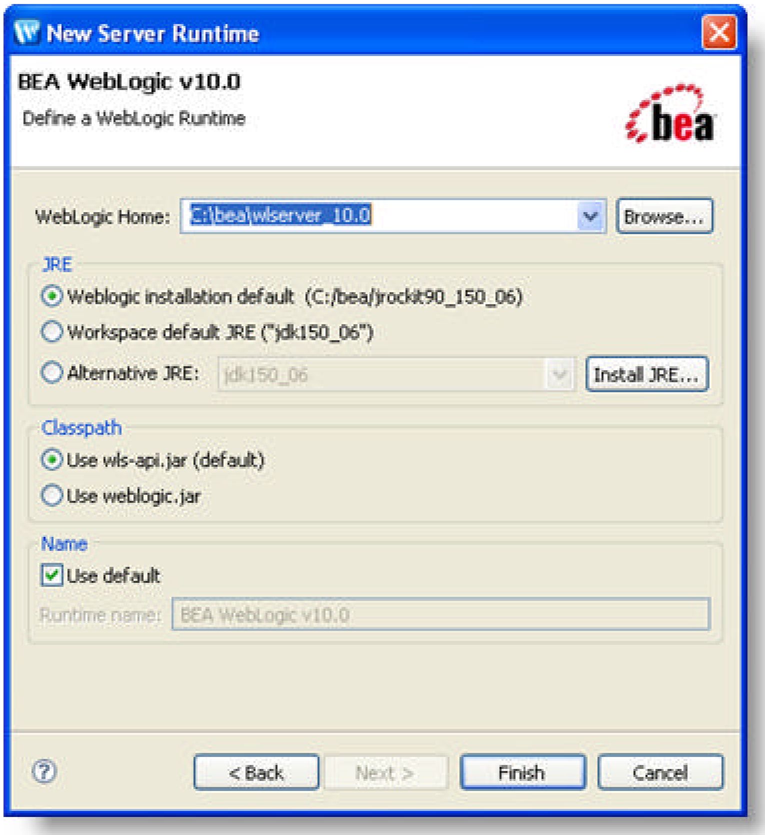
Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.

Installed server runtimes:

Name	Type
BEA WebLogic v10.0	BEA WebLogic v10.0

5. If the BEA WebLogic 10.0 is not installed, select Add.
6. Expand the BEA Systems node and select BEA WebLogic v10.0. Click Next.
7. Browse to the WebLogic installation directory. Click Finish.



New Server Runtime

BEA WebLogic v10.0

Define a WebLogic Runtime

WebLogic Home: C:\bea\wlserver_10.0

JRE

● Weblogic installation default (C:/bea/jrockit90_150_06)

○ Workspace default JRE ("jdk150_06")

○ Alternative JRE: jdk150_06

Classpath

● Use wls-api.jar (default)

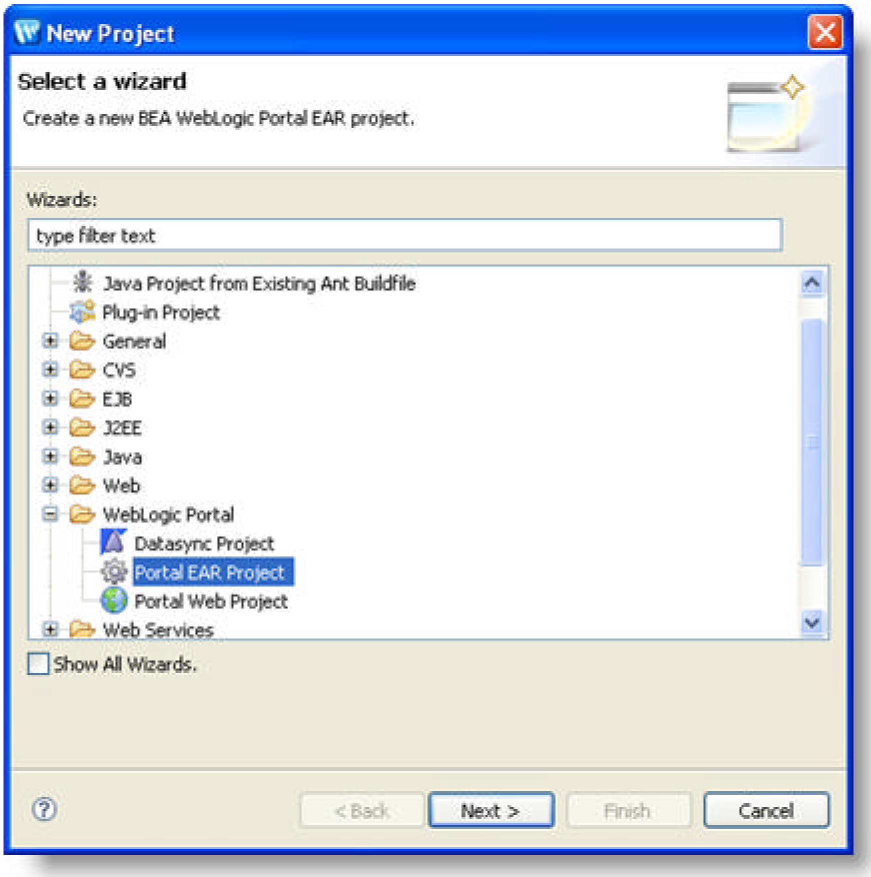
○ Use weblogic.jar

Name

☑ Use default

Runtime name: BEA WebLogic v10.0

8. On the File menu, point to New and click Project.
9. In the Select a wizard page, expand WebLogic Portal and select Portal Ear Project.



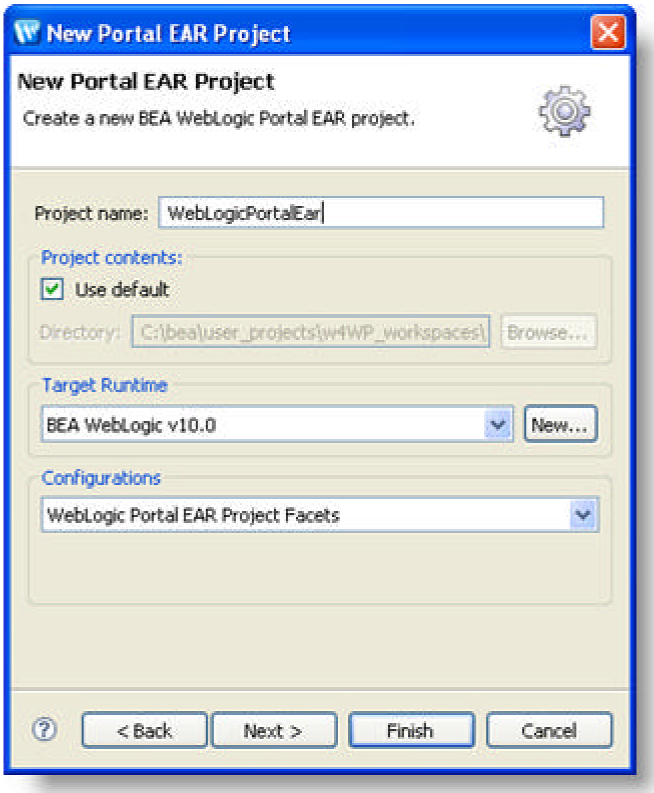
New Project

Select a wizard

Create a new BEA WebLogic Portal EAR project.

10. Click Next.

11. In the New Portal EAR Project page of the wizard, enter a name for the EAR.



New Portal EAR Project

New Portal EAR Project

Create a new BEA WebLogic Portal EAR project.

Project name: WebLogicPortalEar

Project contents:

☑ Use default

Directory: C:\bea\user_projects\w4WP_workspaces\

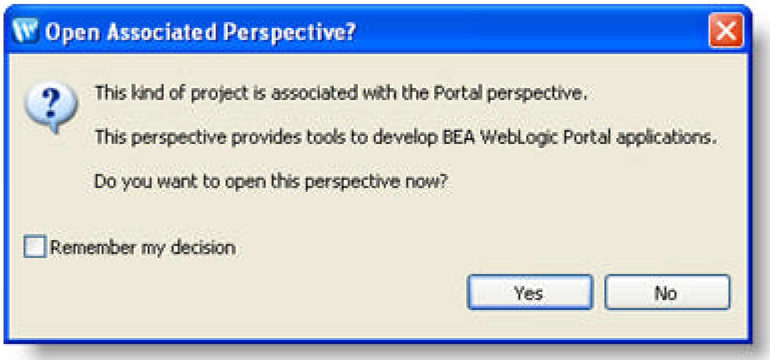
Target Runtime

BEA WebLogic v10.0

Configurations

WebLogic Portal EAR Project Facets

12. Click Finish.
13. In the Open Associated Perspective dialog window that appears, click Yes.



Open Associated Perspective?

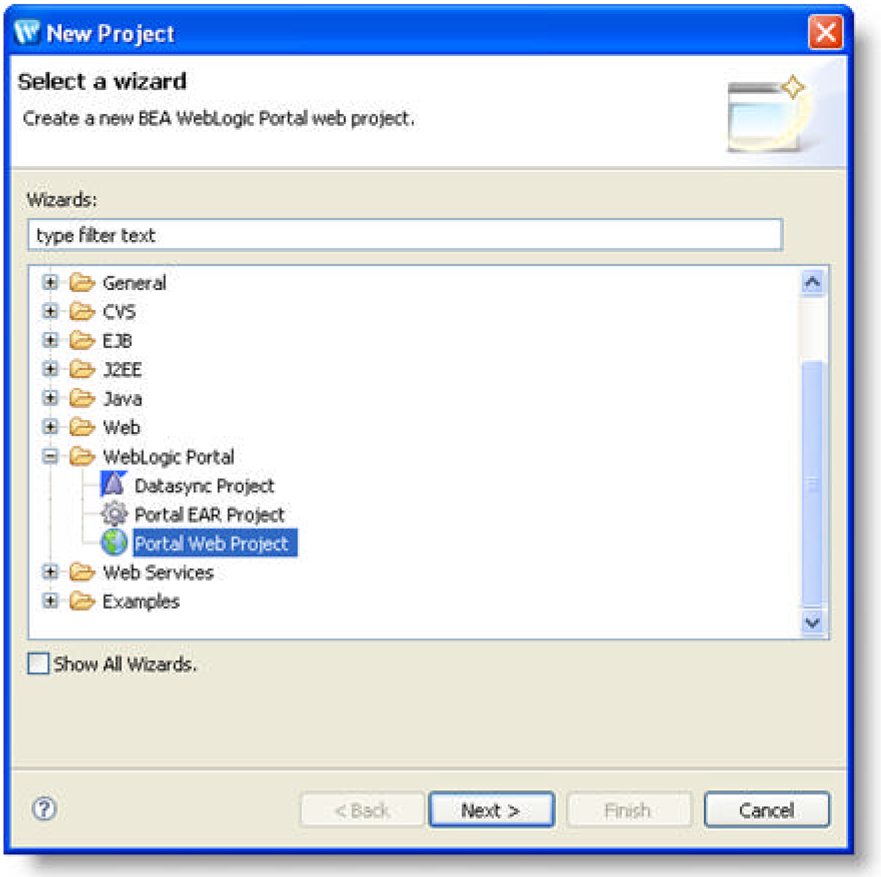
This kind of project is associated with the Portal perspective.

This perspective provides tools to develop BEA WebLogic Portal applications.

Do you want to open this perspective now?

□ Remember my decision

14. On the File menu, point to New and click Project.
15. In the Select a wizard page, expand WebLogic Portal and select Portal Web Project.



New Project

Select a wizard

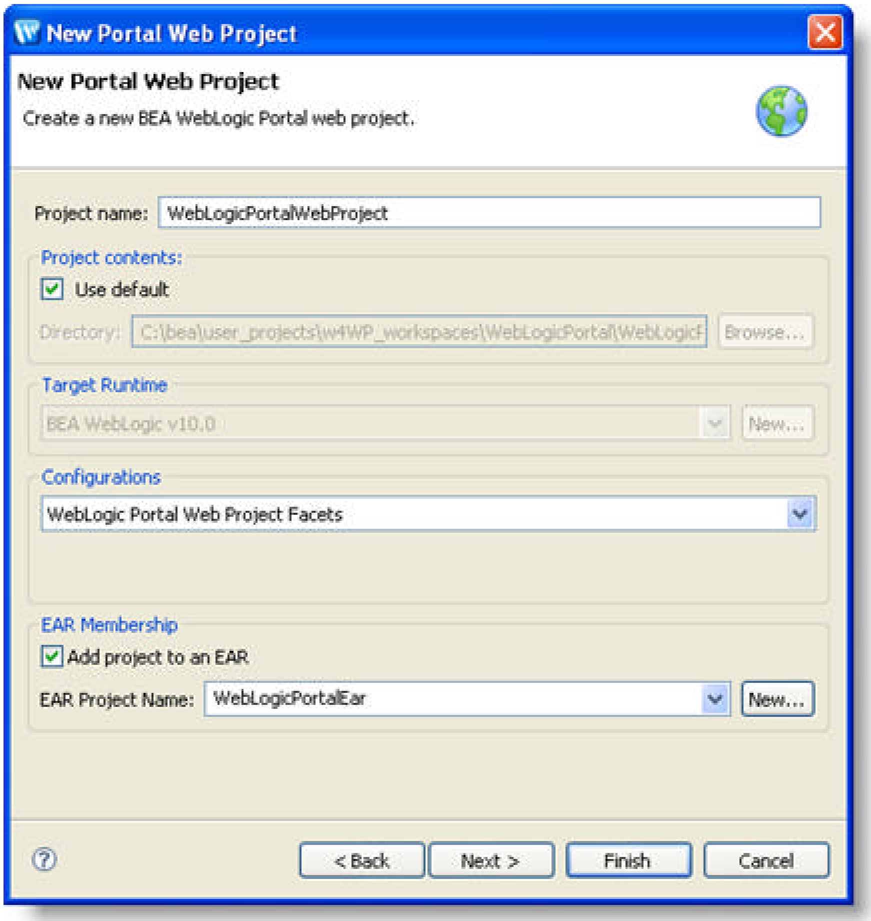
Create a new BEA WebLogic Portal web project.

Wizards:

type filter text

16. In the New Portal Web Project page of the wizard, enter a name for Web Project.

17. Select the Add project to an EAR checkbox. From the Ear Project Name, select the EAR you previously created.



New Portal Web Project

Create a new BEA WebLogic Portal web project.

Project name: WebLogicPortalWebProject

Project contents:

☑ Use default

Directory: C:\bea\user_projects\w4WP_workspaces\WebLogicPortal\WebLogicF

Target Runtime

BEA WebLogic v10.0

Configurations

WebLogic Portal Web Project Facets

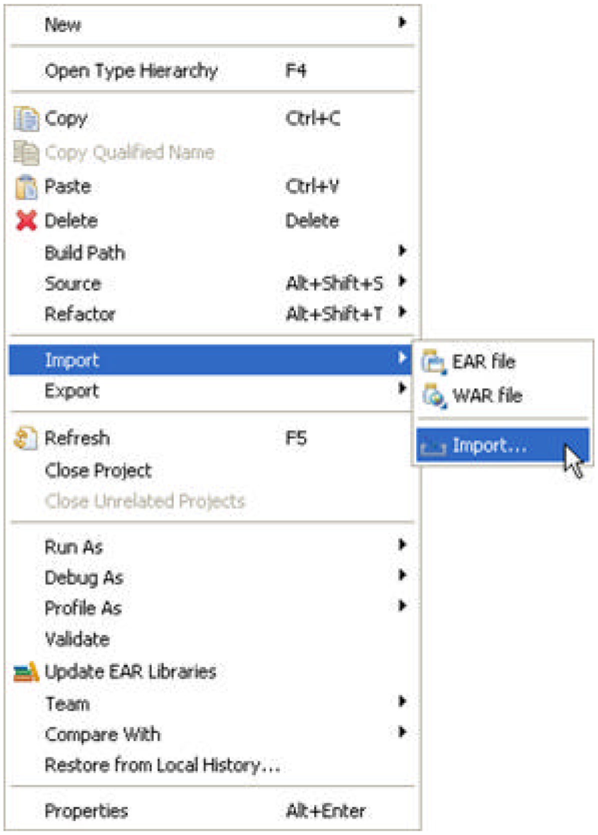
EAR Membership

☑ Add project to an EAR

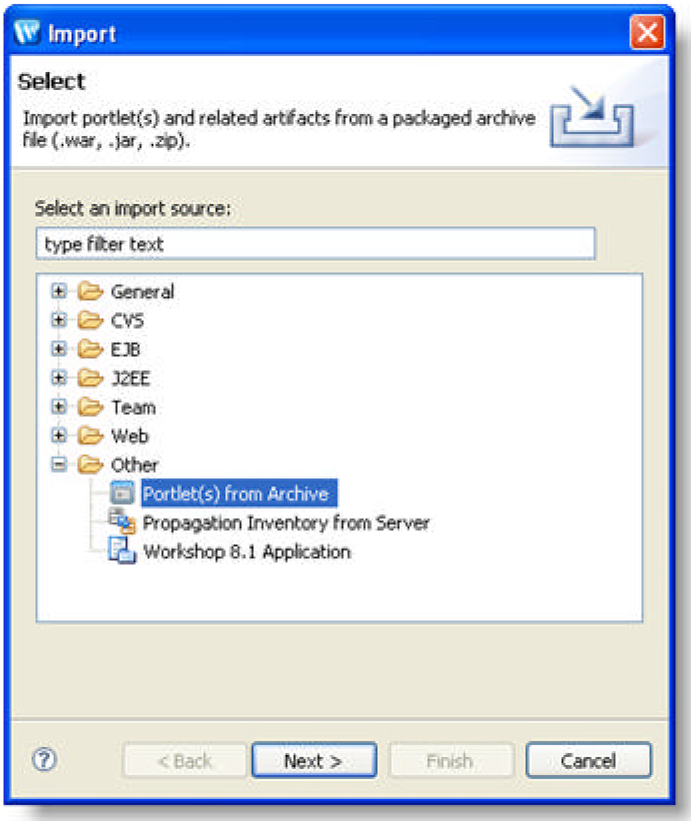
EAR Project Name: WebLogicPortalEar

18. Click Finish.

19. On the Project Explorer, right click your web project, point to Import and click Import...



20. In the Select page of the wizard, expand Other and select Portlet(s) from Archive.



Import

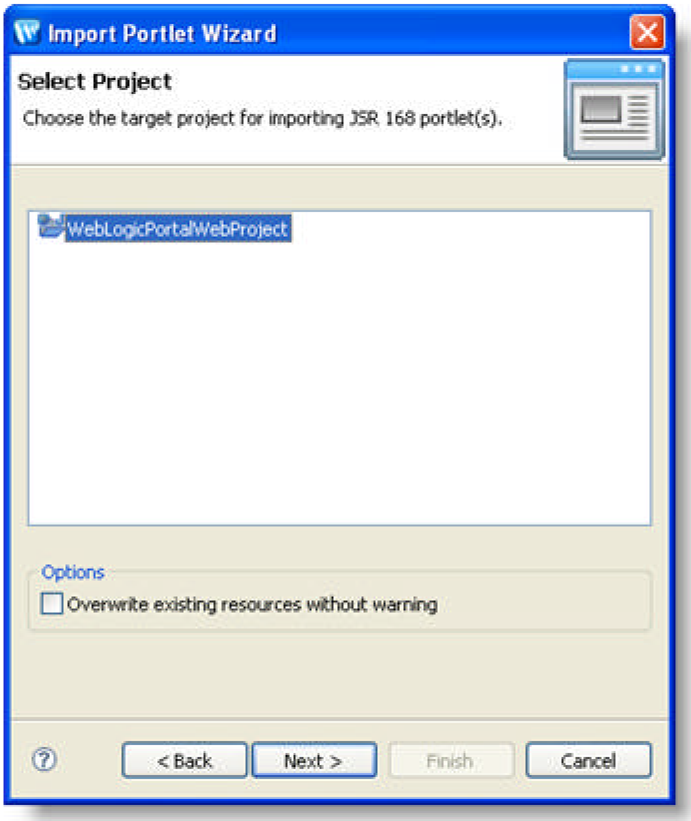
Select

Import portlet(s) and related artifacts from a packaged archive file (.war, .jar, .zip).

Select an import source:

type filter text

21. Click Next.
22. In the Select Project page of the wizard, select your web project as the target to import external portlets to.



Import Portlet Wizard

Select Project

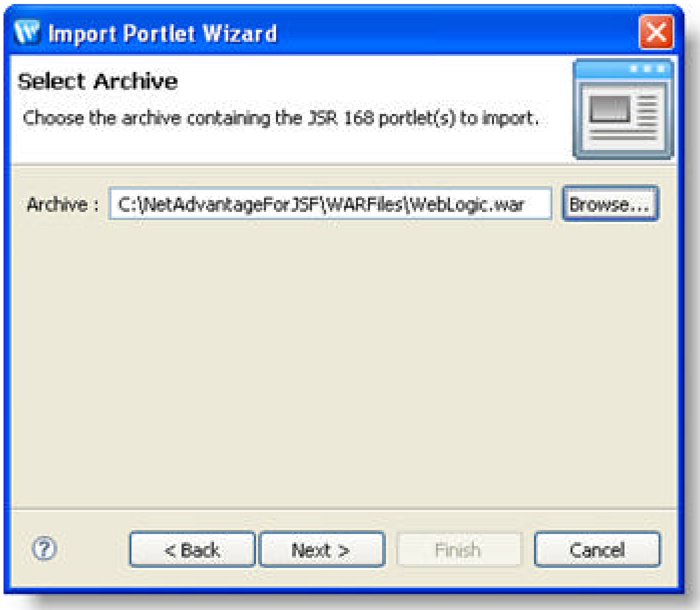
Choose the target project for importing JSR 168 portlet(s).

WebLogicPortalWebProject

Options

□ Overwrite existing resources without warning

23. Click Next.
24. In the Select Archive page of the wizard, browse to the location of your WAR file.



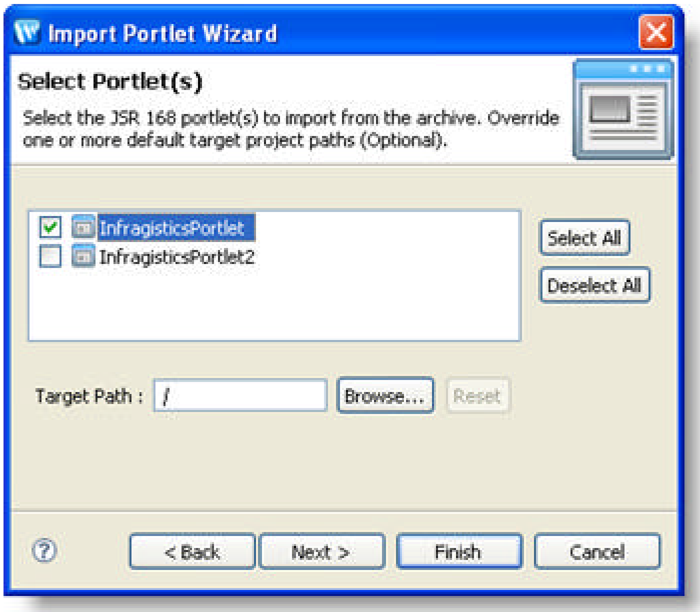
Import Portlet Wizard

Select Archive

Choose the archive containing the JSR 168 portlet(s) to import.

Archive : C:\NetAdvantageForJSF\WARFiles\WebLogic.war

25. Click Next.
26. From the Select Portlet(s) page of the wizard, select InfragisticsPortlet and click Next.



Import Portlet Wizard

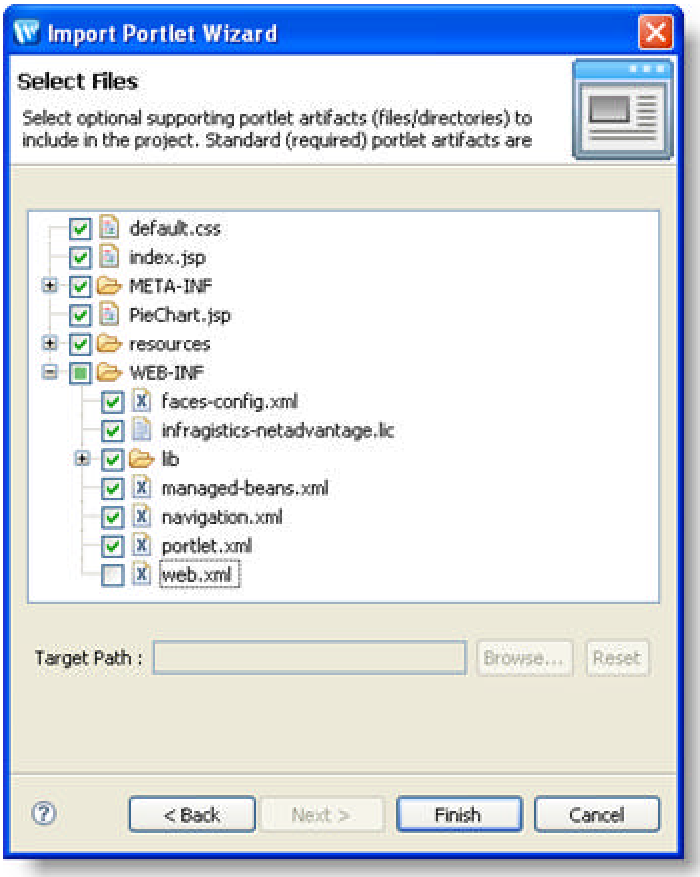
Select Portlet(s)

Select the JSR 168 portlet(s) to import from the archive. Override one or more default target project paths (Optional).

- ☑ InfragisticsPortlet
- □ InfragisticsPortlet2

Target Path : /

27. From the Select Files page of the wizard, select all files, except web.xml.



Import Portlet Wizard

Select Files

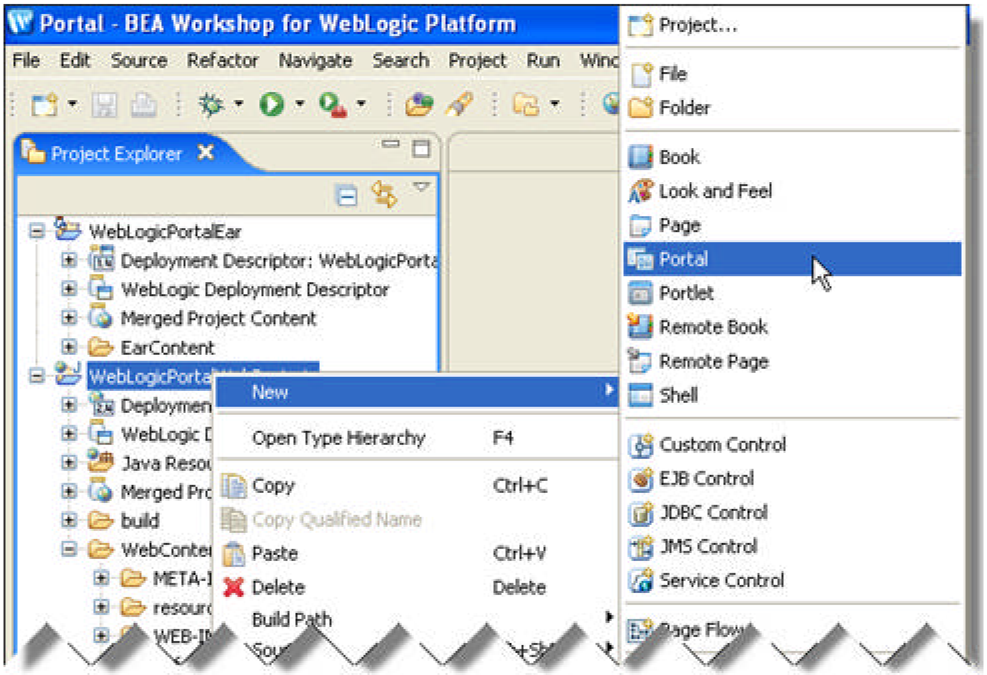
Select optional supporting portlet artifacts (files/directories) to include in the project. Standard (required) portlet artifacts are

- ☑ default.css
- ☑ index.jsp
- ☑ META-INF
- ☑ PieChart.jsp
- ☑ resources
- ▣ WEB-INF
  - ☑ faces-config.xml
  - ☑ infragistics-netadvantage.lic
  - ☑ lib
  - ☑ managed-beans.xml
  - ☑ navigation.xml
  - ☑ portlet.xml
  - □ web.xml

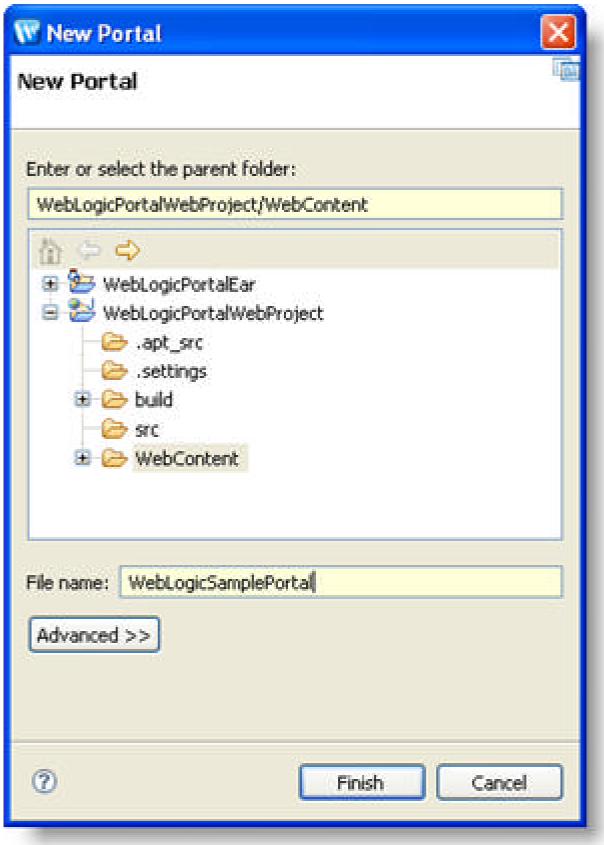
Target Path :

28. Click Finish.

29. On the Project Explorer, right click your web project, point to New and click Portal.



30. From the New Portal page of the wizard, expand your web project and select WebContent as the parent folder. Enter a name for your portal.


New Portal

New Portal

Enter or select the parent folder:

WebLogicPortalWebProject/WebContent

- WebLogicPortalEar
- WebLogicPortalWebProject
  - .apt_src
  - .settings
  - build
  - src
  - WebContent

File name: WebLogicSamplePortal

Advanced >>

Finish Cancel

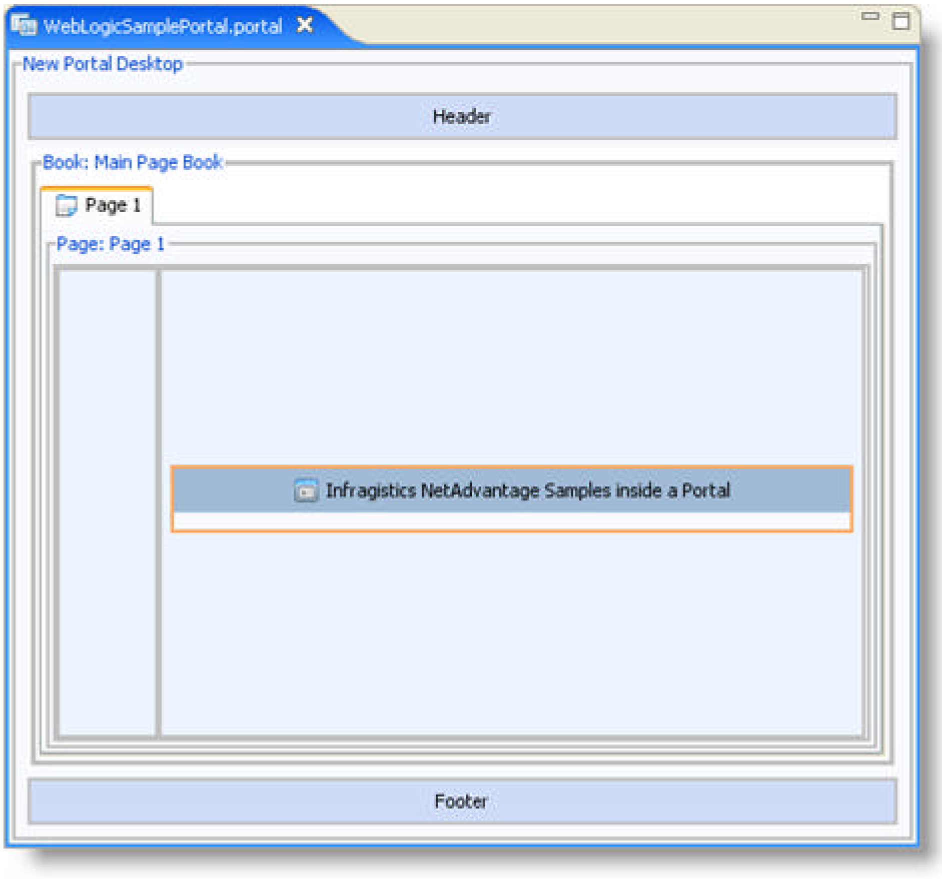
31. Click Finish.

32. Add the following entries to the web.xml file located in WEB-INF folder of your web project
    In XML:

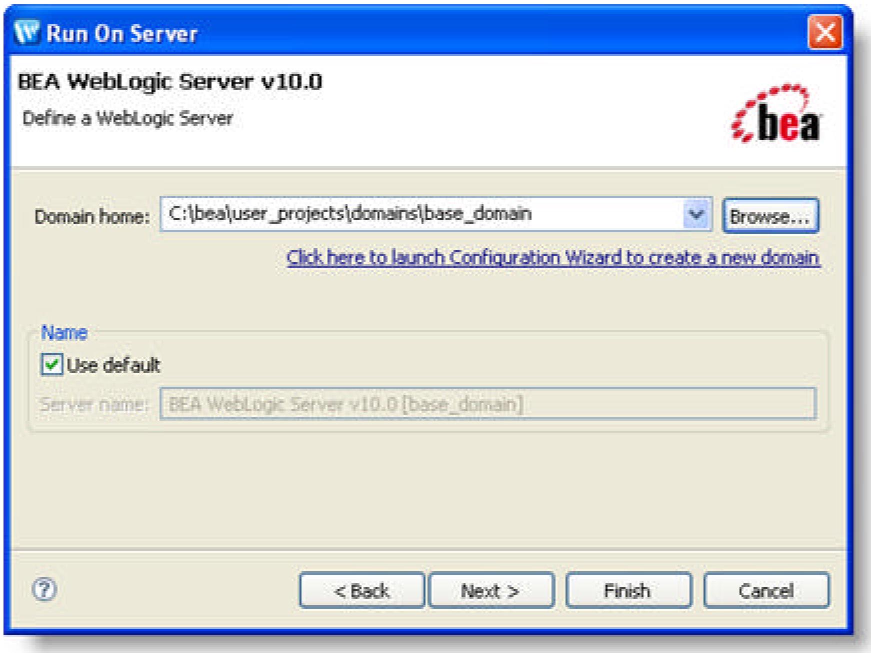
```
<filter>
   <filter-name>IGAjaxFilter</filter-name>
   <filter-class>com.infragistics.faces.portals.WeblogicFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>IGAjaxFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<context-param>
   <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
   <param-value>server</param-value>
</context-param>

<context-param>
   <param-name>javax.faces.CONFIG_FILES</param-name>
   <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
</context-param>

<context-param>
   <param-name>com.infragistics.faces.THEME</param-name>
   <param-value>Claymation</param-value>
</context-param>

<servlet>
   <servlet-name>Faces Servlet</servlet-name>
   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

33. On the Project menu, select Properties. Select Java Build Path, select the Libraries tab. Click Add External JARS... and browse to the igPortalFilter.jar. Click OK. Select Order and Export tab, select igPortalFilter.jar. Click OK.

34. On the Project Explorer, locate the Infragistics portlet and place it on your newly created portal.


WebLogicSamplePortal.portal
New Portal Desktop
Header
Book: Main Page Book
Page 1
Page: Page 1
Infragistics NetAdvantage Samples inside a Portal
Footer

35. Save all your work.
36. On the Project Explorer, right click on your portal definition, point to Run As and select Run On Server.
37. From the Define a New Server page of the wizard, click Next.
38. From the BEA WebLogic Server v10.0, browse to an existing Domain home.



Run On Server

BEA WebLogic Server v10.0

Define a WebLogic Server

Domain home: C:\bea\user_projects\domains\base_domain

Click here to launch Configuration Wizard to create a new domain

Name

☑ Use default

Server name: BEA WebLogic Server v10.0 [base_domain]

39. Click Next.
40. Click Finish.
41. You have successfully run your portlet on a WebLogic portal.

1.4.3 WebSphere Portal

Before you Begin

> **Note:** The files needed for this topic can be found at INSTALL_DIRECTORY\Portal Resources\WebSphere. The jsf-ibm.jar and jsf-portletbridge.jar jars are located in your portal installation directory.

Ensure that:

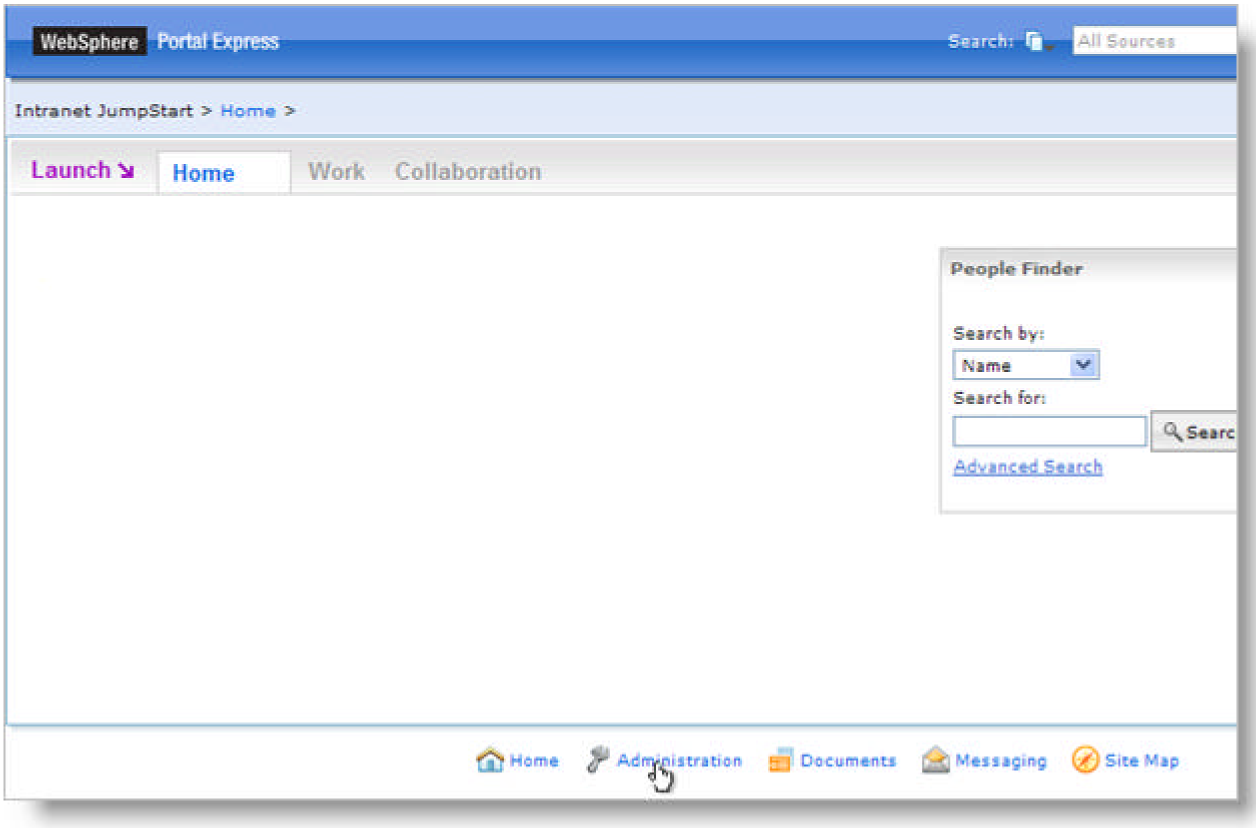
- IBM® WebSphere® Portal or IBM WebSphere Portal Express is installed correctly, for more information on versions, see **Supported Software (Section 1.2)**.
- The server runs without any exceptions.
- You can access http://localhost:<port>/wps/portal
- You can log in to the portal administrative interface with the administrator user name and password.
- You do not have the following in the WEB-INF\lib folder of your web application:
  - jsf implementations such as jsf-api, jsf-impl, myfaces-api and myfaces-impl
  - portletbridge jars
  - portlet.jar
- You have not declared JSF listeners such as com.sun.faces.config.ConfigureListener in your web.xml file.

What You Will Accomplish

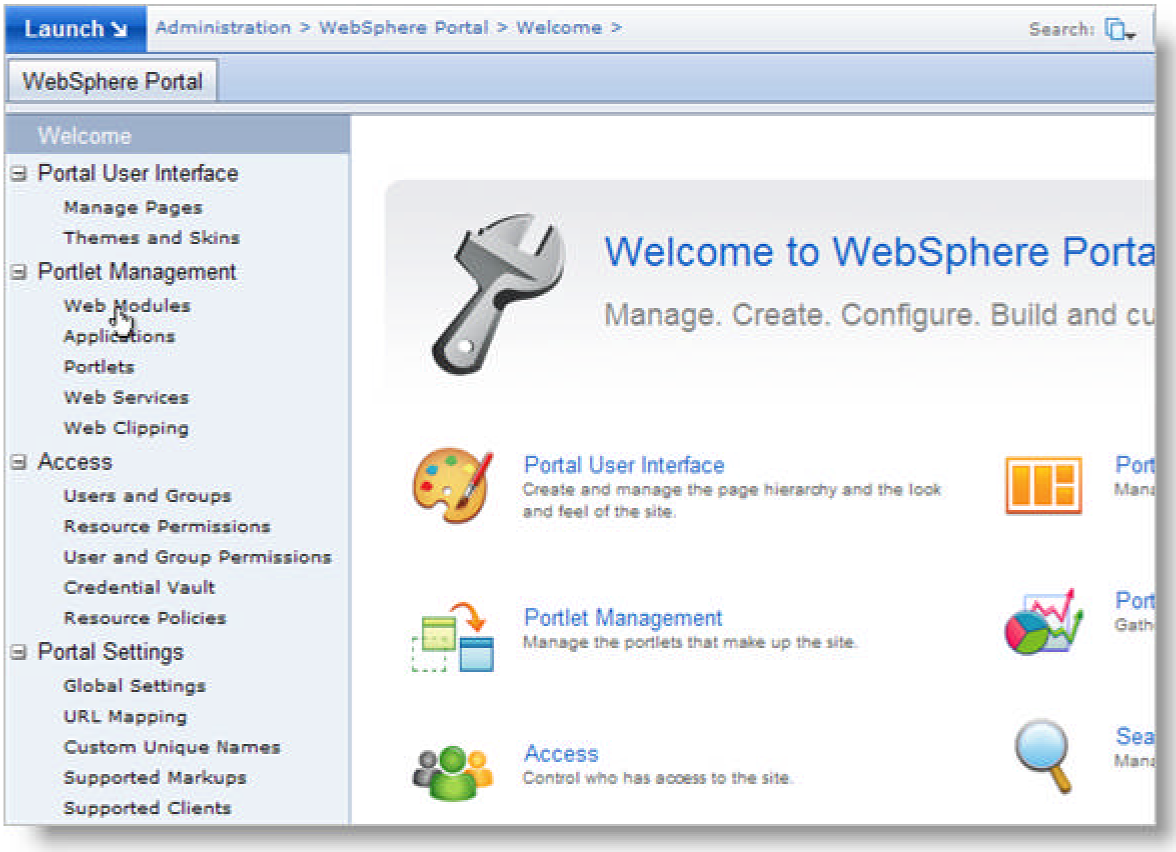
After completing the following steps your JSF components will be aware of the server side filter and all post backs from your JSF components will behave as normal AJAX requests while running in the WebSphere portal.

Follow These Steps

1. Place the portlet-websphere.xml file in your web application's WEB-INF folder and rename it to portlet.xml.
2. Place the following jars into your web application's WEB-INF\lib folder:
   - igPortalFilter.jar
   - jsf-ibm.jar
   - jsf-portletbridge.jar
3. Login to the WebSphere portal home page (http://localhost:<port>/wps/portal) as an administrator .
4. Click Administration, on the bottom of the page.



5. From the table of contents, expand Portal Management and select Web Modules.

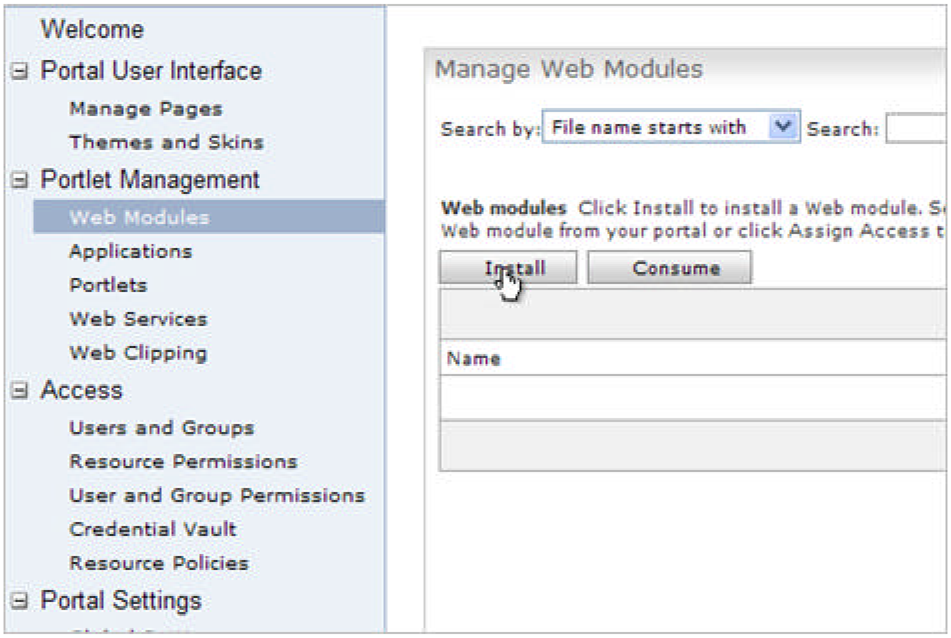


Create and manage the page hierarchy and the look and feel of the site.

Manage the portlets that make up the site.

Control who has access to the site.

6. From the Manage Web Modules page, select Install.

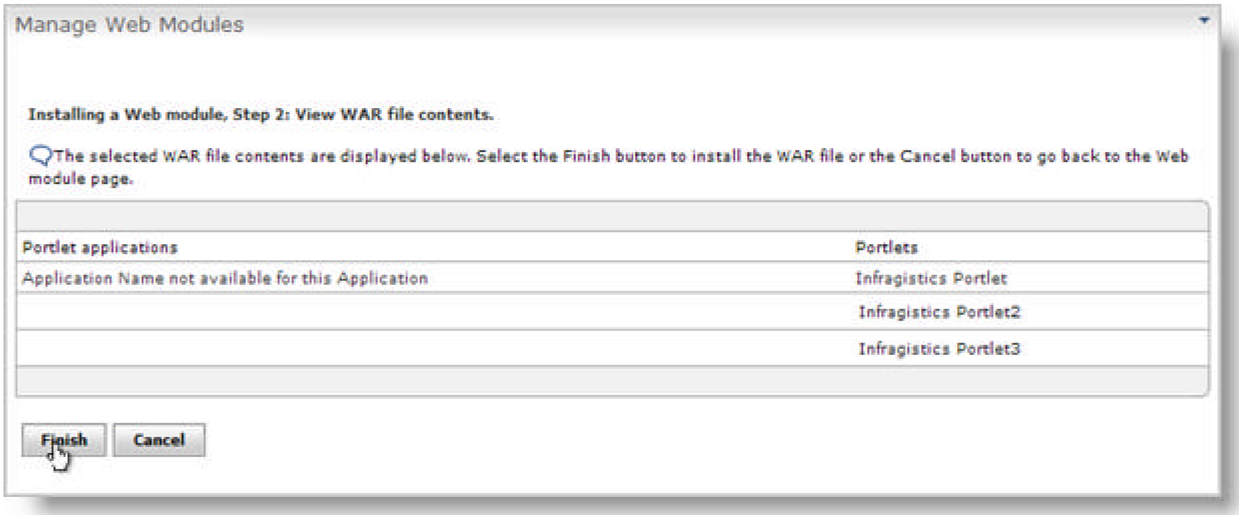


Search by: File name starts with Search:

Web modules Click Install to install a Web module. S
Web module from your portal or click Assign Access t

7. Browse to the location of your web application. Click Next.

8. After reviewing your WAR file contents, click Finish.

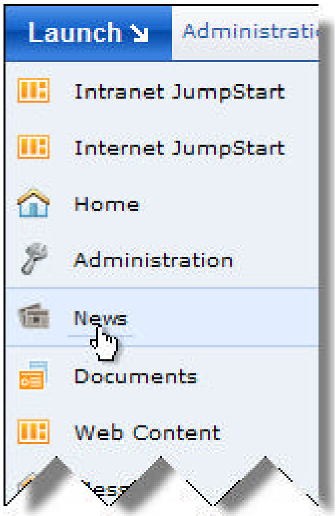

Manage Web Modules

Installing a Web module, Step 2: View WAR file contents.

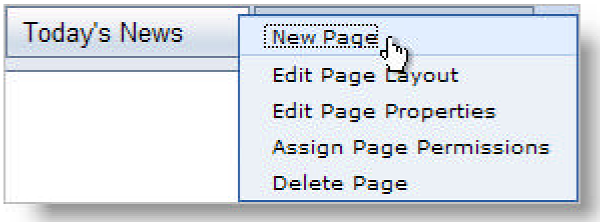
The selected WAR file contents are displayed below. Select the Finish button to install the WAR file or the Cancel button to go back to the Web module page.

Portlet applications	Portlets
Application Name not available for this Application	Infragistics Portlet
	Infragistics Portlet2
	Infragistics Portlet3

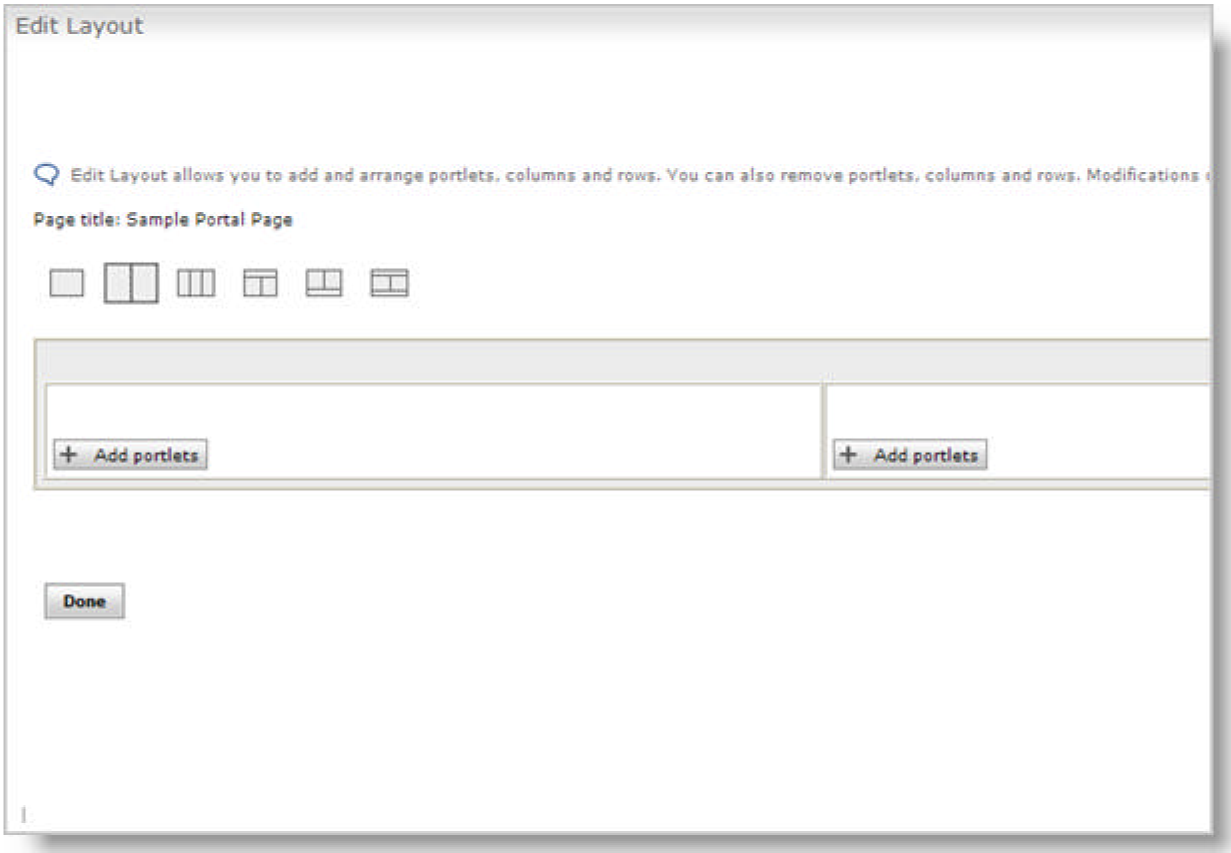
9. On the Launch menu, select News.



10. On the Today's News tab, select New Page.



11. Enter a title for your page. Click OK.
12. From the Edit Layout page, click Add portlets.

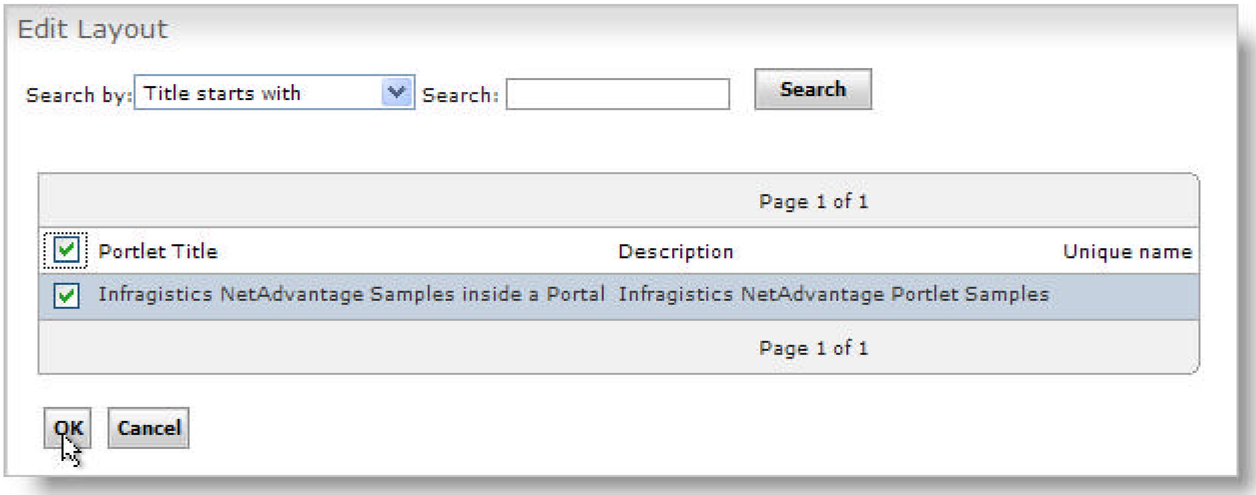


Edit Layout

Edit Layout allows you to add and arrange portlets, columns and rows. You can also remove portlets, columns and rows. Modifications

Page title: Sample Portal Page

13. Select your portlet. Click OK.



Edit Layout

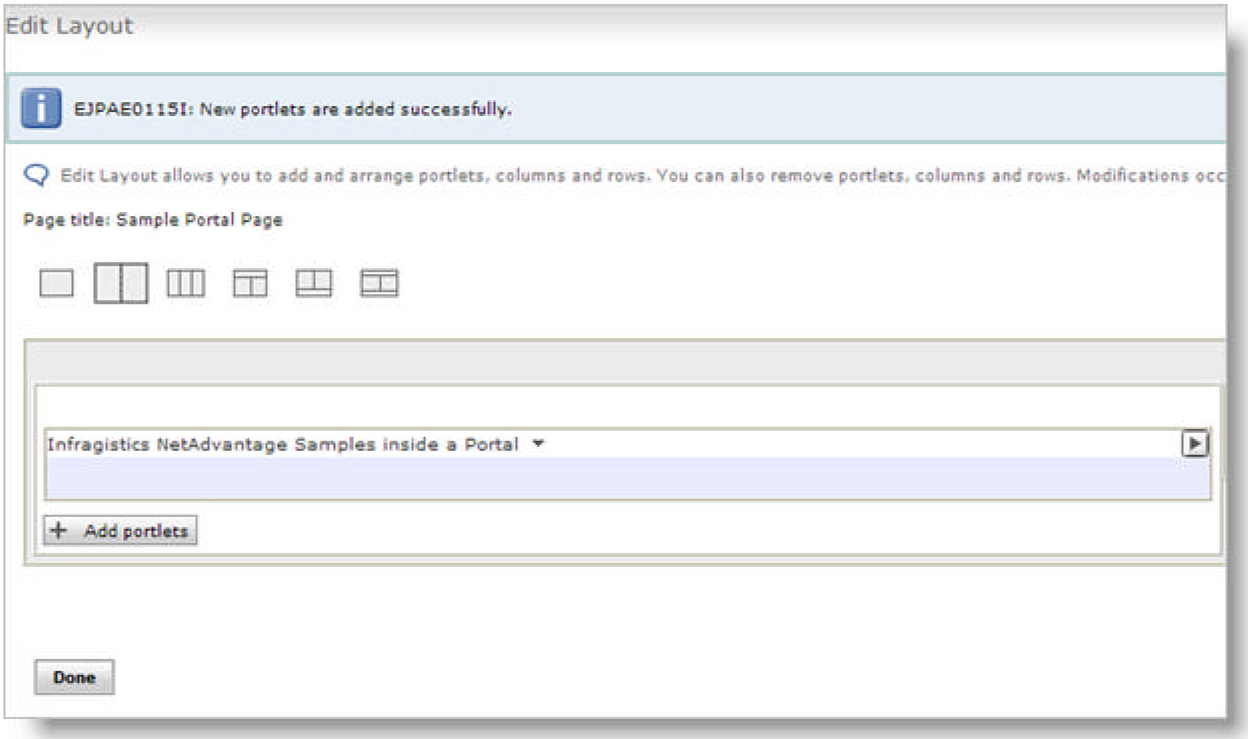
Search by: Title starts with Search:

Page 1 of 1

☑	Portlet Title	Description	Unique name
☑	Infragistics NetAdvantage Samples inside a Portal	Infragistics NetAdvantage Portlet Samples	

Page 1 of 1

14. On the Edit Layout page, you should see a message stating that your portlets was added successfully. Click Done.



Edit Layout

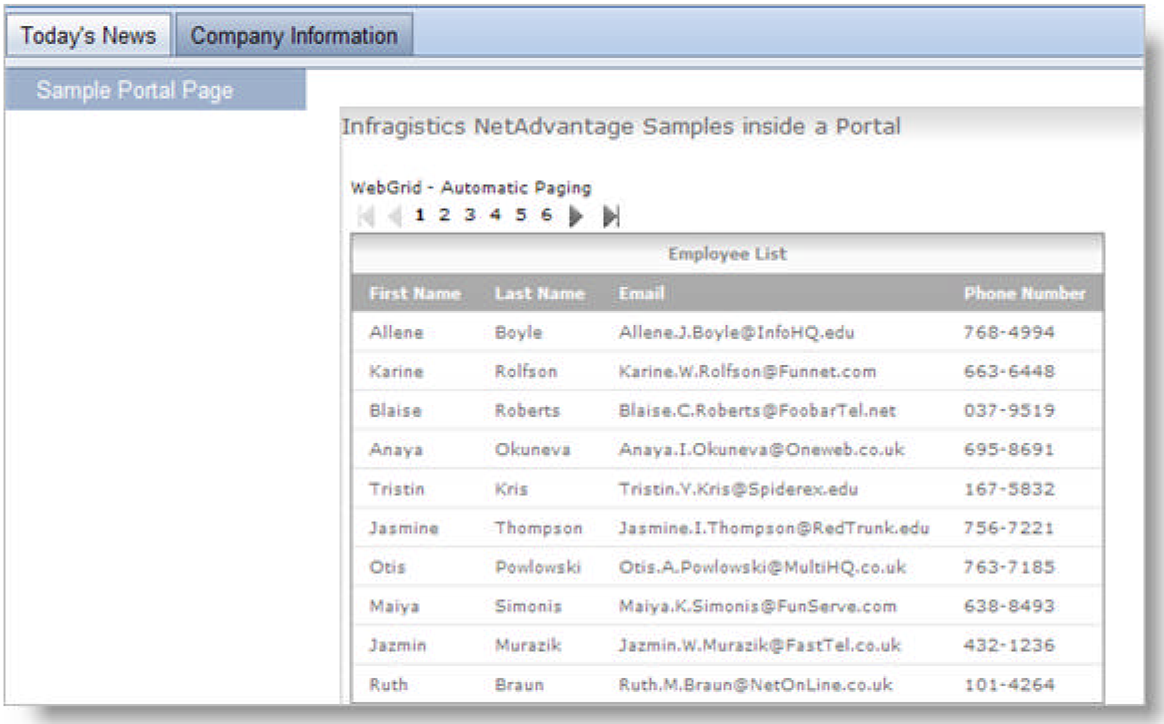
EJPAE0115I: New portlets are added successfully.

Edit Layout allows you to add and arrange portlets, columns and rows. You can also remove portlets, columns and rows. Modifications occ

Page title: Sample Portal Page

Infragistics NetAdvantage Samples inside a Portal

15. You should now see your application running inside the WebSphere portal.



Infragistics NetAdvantage Samples inside a Portal

WebGrid - Automatic Paging

Employee List

First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221
Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493
Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236
Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264

1.5 NetAdvantage for JSF Components, JSF 1.1 and JSF 1.2

This topic describes a few differences between the JSF 1.1 and the JSF 1.2 framework when using NetAdvantage for JSF components.

There is full backward compatibility between JSF 1.2 and JSF 1.1 versions. This means that all your web applications that you created using the JSF 1.1 framework will also run on the JSF 1.2 framework.

JSF 1.2 supports the new Unified Expression Language, which integrates the expression languages defined by JSP 2.0 and JSF 1.1. In the NetAdvantage for JSF 2008 volume 2 - JSF 1.2 support release, the NetAdvantage for JSF components supports the following:

- #{value}
- value

The following code demonstrates how you operate with values and methods in JSF 1.1, JSF 1.2 and also using the UnifiedValueExpression and the UnifiedMethodExpression classes. These are helper classes that allow you to use NetAdvantage for JSF components in your web application independent of the framework.

Operating with Values

JSF 1.1

In Java:

```
FacesContext context = FacesContext.getCurrentInstance();
Application app = context.getApplication();
ValueBinding binding = app.createValueBinding("#{userBean.userid}");
String userid = (String) binding.getValue(context);


// Update this value

binding.setValue(context,"NewUserId");
```

JSF 1.2
In Java:

```
FacesContext context = FacesContext.getCurrentInstance();
ELContext elContext = context.getELContext();
Application app = context.getApplication();
ExpressionFactory factory = app.getExpressionFactory();
ValueExpression ex = factory.createValueExpression(elContext,
"#{userBean.userid}", String.class);
String userId =(String) ex.getValue(elContext);
ex.setValue(elContext, "newUserId");
```

Universal
In Java:

```
FacesContext context = FacesContext.getCurrentInstance();
UnifiedValueExpression uve = UnificationHelper.createUnifiedValueExpression(context,
uve.setValue(context, "newUserId");
```

Operating with Methods

JSF 1.1

In Java:

```
Object result = null;
   FacesContext context = FacesContext.getCurrentInstance();
   Application app = context.getApplication();
   Class[] paramTypes = new Class[] { UserInfo.class, String.Class };
   MethodBinding mb = app.createMethodBinding("#{userBean.addUser}",   paramTypes);
  try
   {
         result = mb.invoke(context, new Object [] {UserInfo(),"joe.dow"});
   }
   catch (EvaluationException e)
   {
         Throwable wrapped = e.getCause();
   }
```

JSF 1.2

In Java:

```
Object result = null;
FacesContext context = FacesContext.getCurrentInstance();
ELContext elContext = context.getELContext();
Application app = context.getApplication();
Class[] paramTypes = new Class[] { UserInfo.class, String.class };
ExpressionFactory factory = app.getExpressionFactory();
MethodExpression me = factory.createMethodExpression(elContext, "#{userBean.addUser}"
try
   {
         result = me.invoke(context, new Object [] {UserInfo(),"joe.dow"});
   }
catch (EvaluationException e)
   {
         Throwable wrapped = e.getCause();
   }
```

Universal

In Java:

```
FacesContext context = FacesContext.getCurrentInstance();
Class[] paramTypes = new Class[] { UserInfo.class, String.class };
UnifiedMethodExpression ume =    UnificationHelper.createUnifiedMethodExpression(cont
ume.invoke(context, new Object [] {UserInfo(),"Allene Boyle"});
```

1.6 Deploying Your Application

This section provides useful information on what you need to do in order to deploy a Web application that uses NetAdvantage for JSF components.

- **Deploying Your Application (Section 1.6.1)**
- **Files and Folders Required for Deployment (Section 1.6.2)**
- **Changing the Location of the Resource Files (Section 1.6.3)**

1.6.1 Deploying Your Application

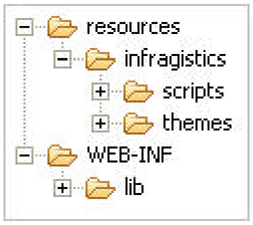
This topic contains important information about how to deploy a Web application that uses our NetAdvantage for JSF components.

To install and deploy the NetAdvantage for JSF components to a web server:

1. Download JavaServer™ Faces, for more information on versions, see **Supported Software (Section 1.2)**, from the **Sun Microsystem's download page (http://java.sun.com/javaee/javaserverfaces/download.html)**, if you do not already have it installed.
2. Copy the following files and paste them into your Web application's WEB-INF/lib folder:
   - jsf-api.jar
   - jsf-impl.jar
3. Copy the JAR files from the INSTALL_DIRECTORY/lib/runtime folder, and paste them into your web application's WEB-INF/lib folder.
4. Verify that your Web application's directory structure is correct, and contains the appropriate Infragistics files and folders. See the "Directory Structure" section below for more information.

Directory Structure

When deploying an application that uses NetAdvantage for JSF components, you must distribute all of the required folders and files in the appropriate directory structure in order for your Web application to function properly. Below is a screen shot of a high-level view of the required directory structure.



> **Note:** By default, the resources folder is located in the WEB_APP folder, where *WEB_APP* is the folder name of your application. You can specify a different location for the resources folder by modifying the *WEB_APP*/conf/web.xml file. For more information, see **Changing the Location of the Resource Files (Section 1.6.3)**.

The list below describes the three types of files that are associated with each NetAdvantage for JSF component group, and need to be deployed at run time in order for the components to function properly. For information on the location of these files, see **Files and Folders Required for Deployment (Section 1.6.2)**.

- JavaScript files – These files are used by the NetAdvantage for JSF components on the client side. The JavaScript files must be deployed to the resources/infragistics/scripts directory.
- Theme files – These files are CSS and image files that are required by the NetAdvantage for JSF components to display the various themes. These files must be deployed to the resources/infragistics/theme/theme_name directory, where theme_name is the name of the Infragistics theme used by the Web application.
- JAR files – These files contain the classes that the NetAdvantage for JSF components depend upon. These files must be deployed to the WEB-INF/lib directory.

All of the above-mentioned files follow a naming convention. Files/folders that are associated with a specific

NetAdvantage for JSF component group contain the name of the component group within the name of the file/folder. For example, in the WEB-INF/lib folder, the infragistics-webgrid.jar file is used only by the WebGrid component group. There are also files/folders that are shared by all the component groups. These files/folders usually contain the name "shared" or "core" in the name of the file/folder. For example, in the WEB-INF/lib folder, the infragistics-shared.jar file is required by all the component groups.

> **Note:** Although it is recommended to deploy all files/folders as they are required by all of the NetAdvantage for JSF components, it is possible to deploy only a subset of the JAR, JavaScript, CSS, and image files. See the table below for a list of the specific files/folders required by each component group.

The table below lists the files/folders that are required by each component group.

Component Group	Required Files		
	Script Folder	Themes Folder	Lib Folder
*Shared	igf_core.js igf_ui.js	igf_shared.CSS /shared folder	infragistics-shared.jar
WebBar	igf_bar.js	igf_bar.CSS /webbar folder	infragistics-webbar.jar
WebChart		igf_chart.CSS /webchart folder	infragistics-webchart.jar
WebDialogWindow	igf_dialogwindow.js	igf_dialogwindow.CSS /webdialogwindow folder	infragistics-webdialogwindow.jar
WebGrid	igf_grid.js	igf_grid.CSS /webgrid folder	infragistics-webgrid.jar
WebInput	igf_input.js	igf_input.CSS /webinput folder	infragistics-webinput.jar
WebMenu	igf_menu.js	igf_menu.CSS /webmenu folder	infragistics-webmenu.jar
WebTab	igf_tab.js	igf_tab.CSS /webtab folder	infragistics-webtab.jar
WebTree	igf_tree.js	igf_tree.CSS /webtree folder	infragistics-webtree.jar

*When deploying your application, you must distribute all files and folders used by the Shared component, regardless of which NetAdvantage for JSF component groups are used by your Web application.

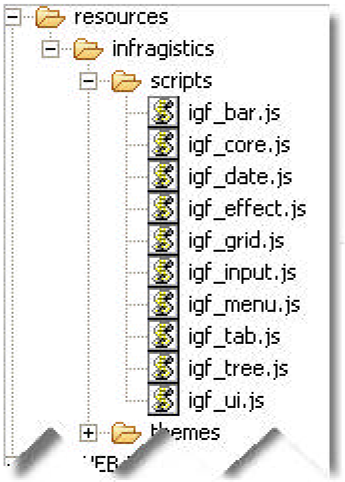
1.6.2 Files and Folders Required for Deployment

This topic describes the folders and files that are required when deploying a Web application that uses NetAdvantage for JSF components.

Note: If you want to move the resource files to a different location, see **Changing the Location of the Resource Files (Section 1.6.3)**.

The portlet.jar should be present in all webapps that you are assembling, except if you are deploying your application in a portal environment, for more information on portals, see **Portal Support (Section 1.4)**. The portlet.jar is located in the INSTALL_DIRECTORY\lib\runtime folder.

scripts Folder



```
resources
  infragistics
    scripts
      igf_bar.js
      igf_core.js
      igf_date.js
      igf_effect.js
      igf_grid.js
      igf_input.js
      igf_menu.js
      igf_tab.js
      igf_tree.js
      igf_ui.js
    themes
```

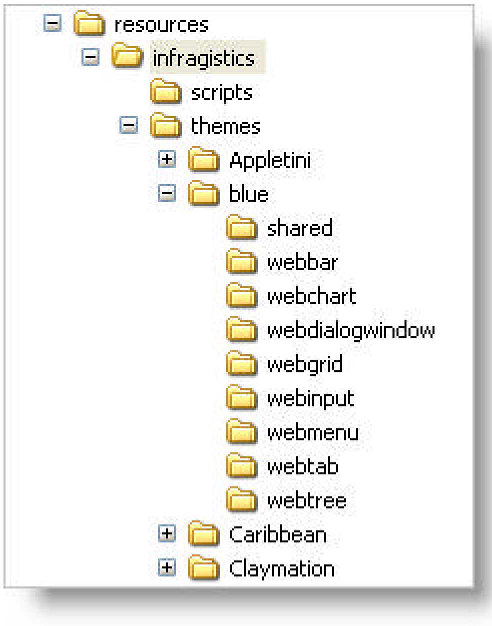
The /resources/infragistics/scripts folder of the deployed application must contain the JavaScript support files required by the NetAdvantage for JSF components. These files can be found in the *install_dir*/lib/runtime/resources/infragistics/scripts folder.

themes Folder

The /resources/infragistics/themes folder of the deployed application must contain the subfolders that are named after the wide variety of themes (e.g., "Pear", "Peach", and "SpringGreen") used by the NetAdvantage for JSF components that are being deployed.

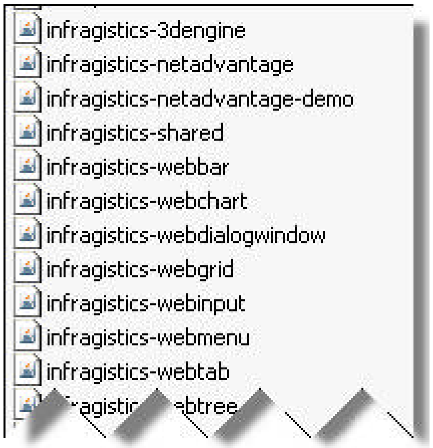
Each theme folder (e.g., "blue") contains a set of CSS files for each of the NetAdvantage for JSF component groups, as well as subfolders (e.g., "webbar" and "webgrid") named after the NetAdvantage for JSF component groups. Each subfolder contains images for the themes supported by the component group.

The theme files can be found in the *install_dir*/lib/runtime/resources/infragistics/themes folder.



- resources
  - infragistics
    - scripts
    - themes
      - Appletini
      - blue
        - shared
        - webbar
        - webchart
        - webdialogwindow
        - webgrid
        - webinput
        - webmenu
        - webtab
        - webtree
      - Caribbean
      - Claymation

lib Folder



- infragistics-3dengine
- infragistics-netadvantage
- infragistics-netadvantage-demo
- infragistics-shared
- infragistics-webbar
- infragistics-webchart
- infragistics-webdialogwindow
- infragistics-webgrid
- infragistics-webinput
- infragistics-webmenu
- infragistics-webtab
- infragistics-webtree

The /WEB-INF/lib folder of the deployed application must contain the Infragistics JAR files. These JAR files contain code that is used by the NetAdvantage for JSF components.

The JAR files can be found in the *install_dir*/lib/runtime folder.

1.6.3 Changing the Location of the Resource Files

The NetAdvantage for JSF product contains various resource files (i.e., icons, cascading style sheets (CSS), and JavaScript files). The default location for these resource files is the *WEB_APP*/resources/infragistics folder, where *WEB_APP* is the folder name of your Web application.

You have the option of moving these resource files to a different location. For example, you may want to move the images to the *WEB_APP*/resources/images folder, the CSS files to the *WEB_APP*/resources/images folder, and the JavaScript files to the *WEB_APP*/resources/scripts folder.

To change the location of the resource files, you need to modify the *WEB_APP*/conf/web.xml file. Using the previously-mentioned example, you would need to modify the web.xml file using the text below, then save the file.

In XML:

```
<context-param>
        <param-name>com.infragistics.faces.RESOURCES_FOLDER</param-name>
        <param-value>/put the location of the resources folder here</param-value>
</context-param>
```

1.7 Installing and Deploying the Samples

A sample application has been built and packaged as the infragistics-netadvantage.war file in the demos/war/j2ee.VERSION directory (where VERSION is the J2EE version that you are using). This application contains all of the NetAdvantage for JSF samples.

> **Note:** You will need to place the license file into the WEB-INF folder of the sample, once it has been deployed. For more information, see **Managing the License (Section 1.8)**.

For information on how to install and deploy the sample application on an application server, click the appropriate link below:

- **Apache™ Tomcat (Section 1.7.1)**
- **BEA WebLogic™ (Section 1.7.2)**
- **IBM WebSphere® (Section 1.7.3)**
- **JBoss® (Section 1.7.4)**
- **Sun Java™ System Application Server (Section 1.7.5)**
- **Facelets Support (Section 1.7.6)**

1.7.1 Apache Tomcat

To install and deploy the sample application on Tomcat:

Note: For more information on supported versions, see **Supported Software (Section 1.2)**.

1. Copy the infragistics-netadvantage.war file to the following directory: <TOMCAT_HOME>/webapps.
2. Start tomcat.
3. The WAR file will begin to deploy. Wait until the deployment process finishes.
4. Open your web browser and enter the following URL to browse the samples:

   http://localhost:*port_number*/infragistics-netadvantage

   where *port_number* is the port on which Tomcat is listening for requests.

1.7.2 BEA WebLogic

To install and deploy the sample application on BEA WebLogic™ 10:

When you are installing and deploying the sample application on BEA WebLogic 10, you will need to use an exploded archive directory. You must manually unpack the infragistics-netadvantage.war file. Our components must be deployed using an exploded archive directory because the *WebChart* component performs a direct file system I/O. For more information on exploded archive directories, see **Using Exploded Archive Directories (http://edocs.bea.com/wls/docs100/deployment/deployunits.html#wp1045820)**.

> **Note:** For more information on supported versions, see **Supported Software (Section 1.2)**.

1. Unpack the infragistics-netadvantage.war file to the following directory: /user_projects/applications.
2. Start your WebLogic application server.
3. Open your Web browser and enter the following URL to browse the samples:

   http://localhost:*port_number*/infragistics**- (http://-/#wp1045820)**netadvantage

   where *port_number* is the port on which WebLogic is listening for requests.

1.7.3 IBM WebSphere

To install and deploy the sample application on IBM WebSphere® 6.1:

> **Note:** For more information on supported versions, see **Supported Software (Section 1.2)**.

1. Start the WebSphere Administrative Console.
2. Double-click the Applications node.
3. Click Install New Application.
4. Click the Browse button, and select the infragistics-netadvantage.war file.
5. In the Context Root text box, type "infragistics-netadvantage".
6. Click Next.
7. Click Next.
8. Click Continue.
9. Click Next.
10. Select the Infragistics sample, then click Next.
11. Select the Infragistics sample, and select the host on which you want to deploy the sample (by default, the default host is selected).
12. Click Next.
13. Click Finish.
14. Click Save to Master Configuration.
15. Click Save.
16. In the left pane, click Enterprise Applications.
17. Select the newly-installed sample, and click Start.
18. Open your Web browser and enter the following URL to browse the samples:

    http://localhost:*port_number*/infragistics-netadvantage

    where *port_number* is the port on which WebSphere is listening for requests.

1.7.4 JBoss

To install and deploy the sample application on JBoss:

> **Note:** For more information on supported versions, see **Supported Software (Section 1.2)**.

1. Remove all the files from the the following directory: <JBOSS4_HOME>/server/default/deploy/jbossweb-tomcat55.sar/jsf-libs.
2. Copy infragistics-netadvantage.war to the following folder: <JBOSS4_HOME>/server/default/deploy.
3. Start your JBoss application server.
4. The WAR file will begin to deploy. Wait until the deployment process finishes.
5. Open your Web browser and enter the following URL to browse the samples:

   http://localhost:*port_number*/infragistics-netadvantage

   where *port_number* is the port on which JBoss is listening for requests.

1.7.5 Sun Java System Application Server

To install and deploy the sample application on Sun Java™ System Application Server 9.1:

> **Note:** For more information on supported versions, see **Supported Software (Section 1.2)**.

1. Copy the infragistics-netadvantage.war file to the following directory: <SUNAS8_HOME>/domains/*your_domain_name*/autodeploy.
2. Start your application server.
3. The WAR file will begin to deploy. Wait until the deployment process finishes.
4. Open your Web browser and enter the following URL to browse the samples:

   http://localhost:*port_number*/infragistics-netadvantage

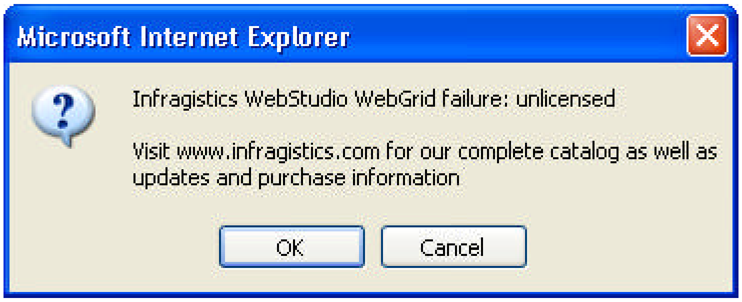
   where *port_number* is the port on which Java System Application Server is listening for requests.

1.7.6 Facelets Support

The NetAdvantage for JSF components should work in a Facelet. This is not server specific, therefore Facelets should work in all the supported application servers. For more information on the supported application servers, see **Supported Software (Section 1.2)**. For an example on using the components in a Facelet see, the **Working with Facelets (Section 7.2.4.7)** tutorial.

1.8 Managing the License

The NetAdvantage for JSF product requires the infragistics-netadvantage.lic file in order to be rendered (e.g., at run time or in a visual IDE). The infragistics-netadvantage.lic file must reside in the WEB-INF directory; otherwise, you will receive the error message below when it displays a page with a NetAdvantage for JSF component. You will also receive this message if you are using an expired license with a newer version of the product.



Microsoft Internet Explorer

Infragistics WebStudio WebGrid failure: unlicensed

Visit www.infragistics.com for our complete catalog as well as updates and purchase information

1.9 Localization

For localization purposes, locale-sensitive data (e.g., error messages, button labels, string literals) are isolated into resource bundles, rather than being hard-coded into the source code. These resource bundles act as a single-source code base so that can be translated into various languages.

Sun Microsystems™ provides an <f:loadBundle> tag, which creates a Load Bundle component that allows you to create a localizable Web application. However, as a result of a known issue with Sun's <f:loadBundle> mechanism, the <f:loadBundle> cannot be used with NetAdvantage for JSF components.

To work around this known issue:

1. Open the faces-config.xml file, and add the following code to the <faces-config> section of the file:
   In XML:

```
<managed-bean>
   <description>BackingBean used to mimic an f:loadBundle tag</description>
   <managed-bean-name>
      msg
   </managed-bean-name>
   <managed-bean-class>
      com.infragistics.faces.shared.helpers.JsfResourceBundle
   </managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
   <managed-property>
      <property-name>baseName</property-name>
      <value>resources.messages</value>
   </managed-property>
</managed-bean>
```

2. Open your .jsp file, and repleace the <f:loadBundle> reference with the following code:
   In JSP:

```
  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
  <h:outputText value="#{msg.message}"/>
```

3. The properties of the .jsp file that contains the message is saved as the following file: META-INF/classes/resources/messages.properties.

The <f:loadBundle> version is as follows:

In JSP:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<h:outputText value="#{msg.message}"/>
```

1.10 Getting Support

If you own a copy of NetAdvantage, you are entitled to certain benefits regarding support services offered by Infragistics.

Creating a Member Profile

Visit the "My Infragistics" area of **Infragistics' Web site (http://www.infragistics.com/)**, and create a **Member Profile (http://www.infragistics.com/scripts/redirects.aspx?id=PROFILE)**. It is important that your most current information is entered into your Member Profile, this will ensure that our Support Service Department can deliver support correspondence and subscription upgrades to you.

Registering your Product Key

After creating a Member Profile, register your Product Key to your MemberID by visiting the **Product Registration (http://www.infragistics.com/scripts/redirects.aspx?id=REGPRDKEY)** page. This will provide you with superior technical support services quickly and conveniently. Your MemberID is connected to your personal profile. Your registration identifies you as the active developer entitled to use and request support for the Infragistics product you have registered.

Getting MemberID Information

You can always visit your **Registration Records (http://www.infragistics.com/scripts/redirects.aspx?id=REGHISTORY)** and review your purchases, issued subscription upgrade keys and registered keys online, 24 hours a day, 7 days a week. After logging in with your MemberID, you'll be able to view vital information including product keys and subscription service and status.

Getting Help

A wealth of valuable information is available at our **DevCenter (http://devcenter.infragistics.com/Default.aspx)**. DevCenter is a customer community location for self-service support information, technical articles, reference applications and much more. For more information on what help is available, hover your mouse over the "Support" menu item in the left pane of DevCenter.

Important **Support Policies (http://www.infragistics.com/scripts/redirects.aspx?id=SUPPORTPOLICY)** and phone support contact information is also available on DevCenter.

Note: You must have valid priority support to access phone support.

Each Infragistics product registered user can submit requests for technical support via the Web site. To log an incident or start any support issue, please log on to DevCenter to **Submit a Support Issue (http://www.infragistics.com/scripts/redirects.aspx?id=SUPPORTISSUE)** and enter your MemberID. For your convenience, you can also attach a sample project if applicable.

Product Life Cycle

Infragistics Product Life Cycle information is available in PDF format and provides the following information:

- name and version of the Infragistics tools
- product maintenance (bug fix) period
- product support period
- product status

Click **here (http://download.infragistics.com/download/pubs/ProductLifeCycle.pdf)** to download the PDF file.

For issues related to your account, product ownership, or registration, please e-mail **Registrations@Infragistics.com**. Include your name and product key (if known) so that we may better assist you. You can also call (609) 448-2000 9 a.m. to 5 p.m., Monday to Friday, EST US.

2 Known Issues and Breaking Changes

The following links provide information on known issues and breaking changes that impact the entire NetAdvantage for JSF product. Please take note of these issues/breaking changes as you use NetAdvantage for JSF.

If an issue arises after a volume is released, you can visit our **forums (http://forums.infragistics.com/forums/209.aspx)** for the latest breaking changes and known issues.

- **Known Issues and Breaking Changes in 2008 Volume 2 (Section 2.1)**
- **Known Issues and Breaking Changes in 2008 Volume 1 (Section 2.2)**
- **Known Issues and Breaking Changes in 2007 Volume 2 (Section 2.3)**

2.1 Known Issues and Breaking Changes in 2008 Volume 2

The following is a list of known issues we became aware of during the NetAdvantage for 2008 volume 2 release. For known issues and breaking changes in previous releases, see **Known Issues and Breaking Changes (Section 2)**.

Known Issue with JSF 1.2 and Sun Application Server 9.1

The NetAdvantage for JSF 2008 volume 2 with support for JSF 1.2 release does not work in Sun® Application Server 9.1.

Known Issue with JSF 1.2.08

There is a known issue with NetAdvantage for JSF components running in the JSF 1.2.08 framework. An issue occurs when the summary length of all attributes exceeds one buffer size in the implementation. This mainly effects the WebChart component and some WebInput components.

2.2 Known Issues and Breaking Changes in 2008 Volume 1

Known Issues of Facelets Support

- The Facelets environment expects a DataModel instance to be bound for the dataSource attribute of a component. For example, if you have the following getter method:
  In Java:

  ```
  public List getTreeNodes() {
     if (treeNodes==null) treeNodes = new java.util.ArrayList();
        return treeNodes;
  }
  ```

  You should change it to use DataModel instead, as demonstrated by the following code:

  In Java:

  ```
  public DataModel getTreeNodes() {
     if (treeNodes==null) treeNodes = new java.util.ArrayList();
        return new ListDataModel(treeNodes);
  }
  ```

- When using value binding, the getter method should not receive parameters
- There is a current limitation in the Facelets environment that when using method binding, you can not pass listeners as parameters in composition templates.
- An exception occurs when running the WebChart component within the facelets environment. This exception occurs because some of the WebChart attributes are of type Number and when Facelets tries to create a concrete number implementation for these attributes, it is unable to do so and throws a "Cannot convert type class.java.lang.String to class java.lang.Number" exception.

Known Issues of WebBar

When WebGrid is nested within a Stackbar or Sidebar component, the content of the WebGrid component does not appear. This issue does not occur with a non-scrolling WebGrid.

Known Issues of WebGrid

- Column resizing is not retained while paging. If you resize the columns on your WebGrid and then navigate to another page of your WebGrid, the columns will revert back to the default size.
- Vertical and Horizontal scroll bars are not shown on the child grid in a hierarchical resizable WebGrid. If the child grid exceeds the width and/or height of the parent grid, you will not be able to scroll horizontally and/or vertically to see the data of the child grid.
- When you set the resizableColumns property to true, you must also set the width attribute in the style property. The following code example demonstrates how to do this.

  In JSP:

  ```
     <ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}">
          pageSize="10"
          fixedColumnCount="2"
          resizableColumns="true"
          style="width: 600px">
  ```

- When you are using the design surface of IBM® Rational® Application Developer for WebSphere 7.0, dragging the ColumnSelectRow component on to the GridView component creates an instance of the

CheckBox component outside of the GridView component. An instance of the ColumnSelectRow component is also created on the GridView component.

- When you are using the design surface of RAD, deleting the Column's header of your WebGrid, using the Delete key, does not successfully delete it.
- When you are using the design surface of RAD, dragging multiple instances of the GridActivation component and the GridEditing component on to your WebGrid produces many unnecessary tags in the source.
- When you are using the design surface of NetBeans™, the last column header of your WebGrid does not extend to your WebGrid's outline. When you are using the design surface of NetBeans, deleting your WebGrid's header, footer, column header and column footer, using the Delete key, does not refresh the Design page automatically.
- When you are running your project from NetBeans, using the Apache™ Tomcat bundled server, the sorting functionality on WebGrid does not work.

Known Issues of WebTab

When you are using the design surface of IBM® Rational® Application Developer for WebSphere 7.0, two or more TabItem's Selected property can be set to true at the same time.

Known Issues of WebTree

When you run your WebTree component, each node in your WebTree is automatically generated as a link.

2.3 Known Issues and Breaking Changes in 2007 Volume 2

We recommend that when you are using Microsoft® Internet Explorer 6 and later, you include the <DOCTYPE> tag in your JSP page. This reduces the risk of your components rendering incorrectly within your application.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Known Issues in WebGrid

- When your data source is configured to use a ResultSet Data Object Model, paging will not work on your WebGrid. It does not work because the following method

  In Java:

  ```
  getRowCount()
  ```

  returns -1. The number of rows in the ResultSet cannot be determined without scrolling through the entire ResultSet which is not feasible if the ResultSet contains many rows.

- When you set the resizableColumns property to true, you must also set the width attribute in the style property. The following code example demonstrates how to do this.

In JSP:

```
   <ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
       pageSize="10"
       fixedColumnCount="2"
       resizableColumns="true"
       style= "width: 600px">
```

Breaking changes in WebChart

The following methods return type have been changed from java.util.List to java.util.Collection:

- GetAxes()
- GetLights()
- GetSeries()

The following methods parameter type have been changed from java.util.List to java.util.Collection:

- SetAxes()
- SetLights()
- SetSeries()

3 What's New

What's New in 2009 Volume 1 (Section 3.1)

Find out which new components and features were added to this volume release of NetAdvantage for JSF.

Revision History (Section 3.2)

Need to find out when a certain feature was added to our controls? Easily track down any new feature since our 2006 Volume 2 release.

NetAdvantage for JSF

3.1 What's New in 2009 Volume 1

The NetAdvantage for JSF 2009 Volume 1 includes a number of powerful new features and functionalities to allow you to take even more advantage of our JSF components.

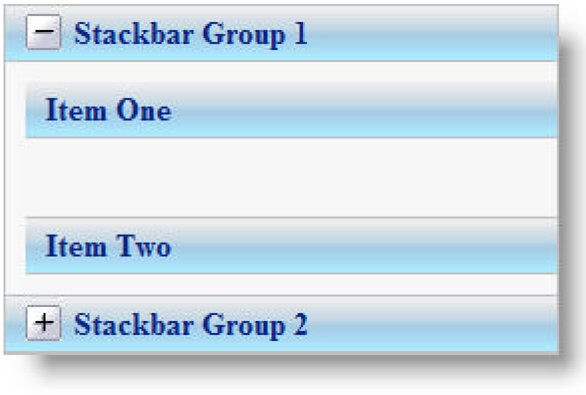
Click the links to see a list of the features being offered:

- **Expanded Icon Added to WebBar (Section 3.1.1)**
- **New Client-Side Object Model (Section 3.1.2)**
- **New Methods Added to SelectedRowsChangeEvent (Section 3.1.3)**
- **You Can Now Change the Default Setting of ColumnSelectRow (Section 3.1.4)**
- **You Can Now Expand or Collapse WebMenu with Mouse Events (Section 3.1.5)**
- **You Can Now Export an Entire WebGrid to CSV (Section 3.1.6)**
- **You Can Now Force a Vertical Scroll Bar (Section 3.1.7)**
- **You Can Now Set the Text of DateChooser Submit Button (Section 3.1.8)**
- **You Can Now Sort a Localized WebGrid (Section 3.1.9)**

3.1.1 Expanded Icon Added to WebBar

We have added a new attribute, expandedIconUrl, to the <ig:stackbarGroup> tag. This allows you to set an icon that is displayed to your end user when the stack bar is expanded.

The existing attribute, iconUrl, is now used to indicate the collapsed state of the stack bar.



Related Topic

Set the Expanded Icon (Section 4.4.2)

3.1.2 New Client-Side Object Model

Starting in the NetAdvantage for JSF 2009 volume 1 release, a new JavaScript-based object model, also known as a Client-Side Object Model (CSOM), is now available on the client for each of our NetAdvantage for JSF components. This CSOM lets you build rich, responsive, Web applications which keep post-backs to a minimum.

Each and every component now has a new tag in the form <ig:*nnnn*ClientEvents> where *nnnn* is the name of the component and you can assign your event listeners declaratively through these Client Event settings.

Related Topic

Getting Started with the Client Side Object Model (Section 4.3)

3.1.3 New Methods Added to SelectedRowsChangeEvent

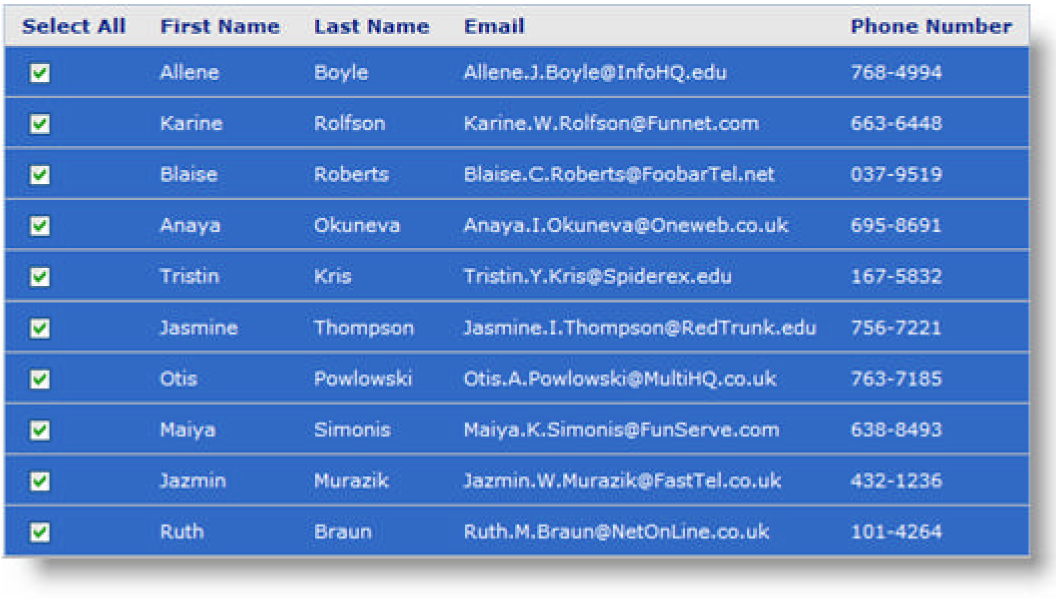
In the NetAdvantage for JSF 2009 volume 1 release, new functionality has been added to the SelectedRowsChangeEvent event. With this new feature you can determine if the SelectAll checkbox is selected and also if the event was triggered from the SelectAll checkbox or from select row checkbox.

Related Topic

SelectedRowsChangeEvent (Section 4.7.3.5.8)

3.1.4 You Can Now Change the Default Setting of ColumnSelectRow

Previously, when using the ColumnSelectRow component in your WebGrid you could not override the default setting. The checkboxes were unselected when they were initially loaded. However, we have added a new feature which allows the checkboxes to be selected when they are initially loaded. You can override the default setting by simply setting the newly added defaultSelected attribute to True in the <ig:columnSelectRow> tag.



Select All	First Name	Last Name	Email	Phone Number
☑	Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
☑	Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
☑	Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
☑	Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
☑	Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
☑	Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221
☑	Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
☑	Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493
☑	Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236
☑	Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264

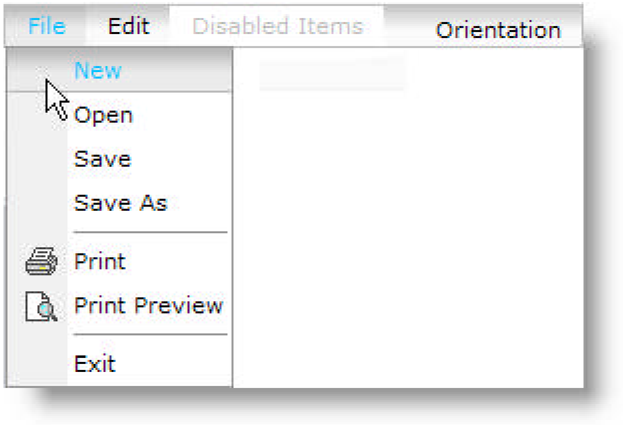
Related Topic

Change the Default Setting of ColumnSelectRow (Section 4.7.3.5.1)

3.1.5 You Can Now Expand or Collapse WebMenu with Mouse Events

Previously, you were not able to control how a menu item opened or closed. However, in the NetAdvantage for JSF 2009 volume 1 release, we have added two new attributes to the <ig:menu> tag, named expandOn and collapseOn. These attributes allow you to specify how the menu item will open or close, through a series of mouse events.

You can set these attributes at the parent level, so that all the children menu items will inherit the setting or you can set it for each individual menu item.

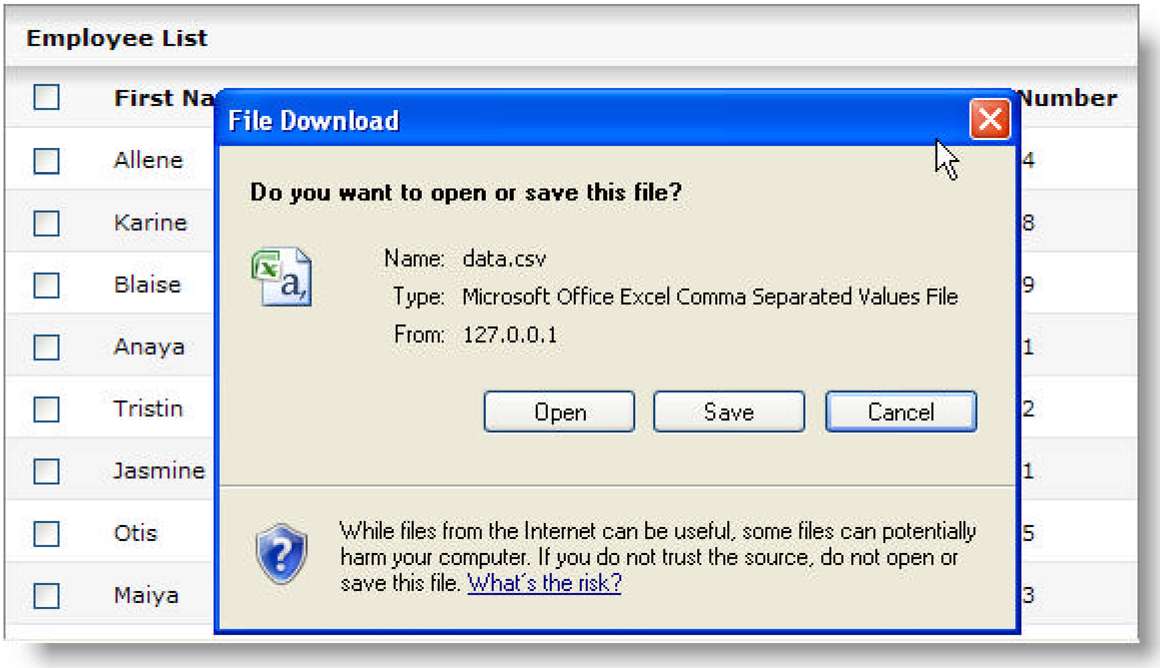


Related Topic

Expand or Collapse WebMenu with Mouse Events (Section 4.9.3)

3.1.6 You Can Now Export an Entire WebGrid to CSV

You could previously export selected rows and the current page of your WebGrid to a Comma Separated Value file. For this release, we have expanded on that functionality and you can now export your entire WebGrid to a CVS file.



Employee List

□	First Na	Number
□	Allene	4
□	Karine	8
□	Blaise	9
□	Anaya	1
□	Tristin	2
□	Jasmine	1
□	Otis	5
□	Maiya	3

File Download

Do you want to open or save this file?

Name: data.csv

Type: Microsoft Office Excel Comma Separated Values File

From: 127.0.0.1

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

Related Topic

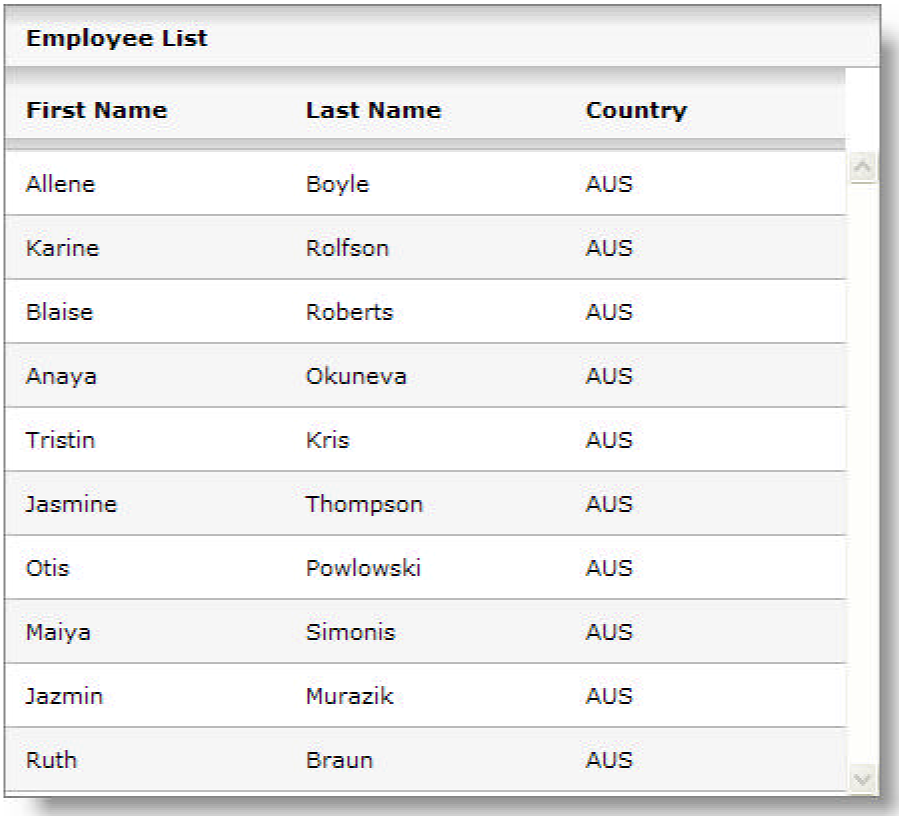
Exporting Data to a CSV File (Section 4.7.3.1.4)

3.1.7 You Can Now Force a Vertical Scroll Bar

Previously vertical scroll bars would only appear on your grid if the content of the grid exceeded the specified height. If the height attribute of the <ig:gridView> is not set, then the vertical scroll bar would not appear.

Starting in the 2009 volume 1 release, we have added a new attribute, forceVerticalScrollBar, to the <ig:gridView> tag.

This allows you to apply vertical scrollbars to your WebGrid component regardless of the value of the height attribute.



Employee List

First Name	Last Name	Country
Allene	Boyle	AUS
Karine	Rolfson	AUS
Blaise	Roberts	AUS
Anaya	Okuneva	AUS
Tristin	Kris	AUS
Jasmine	Thompson	AUS
Otis	Powlowski	AUS
Maiya	Simonis	AUS
Jazmin	Murazik	AUS
Ruth	Braun	AUS

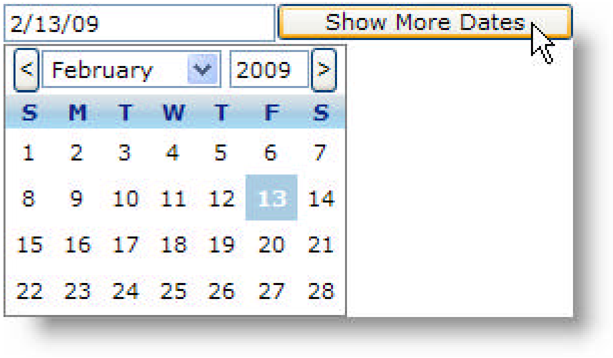
Related Topic

Force Vertical Scroll Bars (Section 4.7.3.3.7)

3.1.8 You Can Now Set the Text of DateChooser Submit Button

Previously you were not able to edit the text of the DateChooser's submit button. However, we've now added functionality that allows you to override the default setting of "...".

To set the text of the button, you simply set the new buttonText attribute of the <ig:dateChooser> to the text value you want to appear.



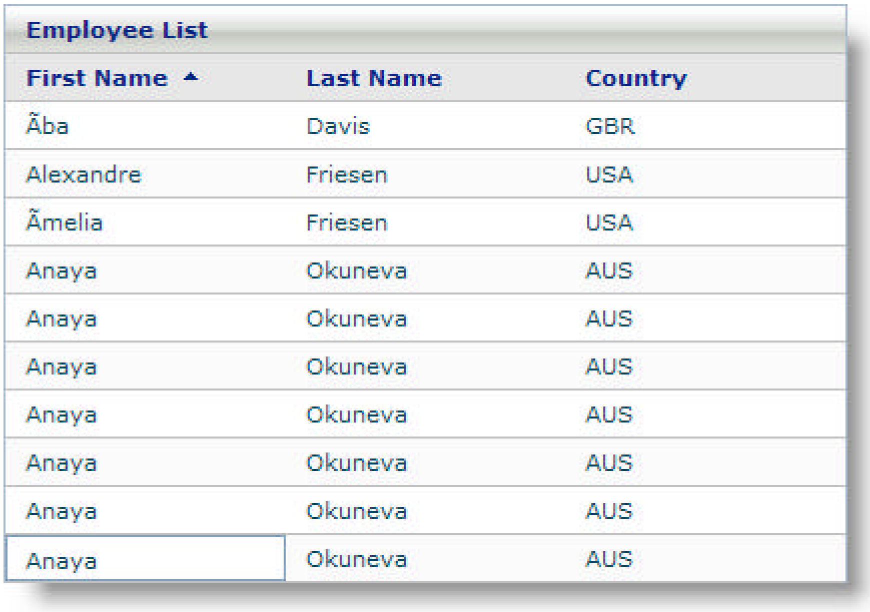
Related Topic

Set the Text of DateChooser Submit Button (Section 4.8.1.3.1)

3.1.9 You Can Now Sort a Localized WebGrid

Sorting has been a feature of our WebGrid component for a long time; in this release we have expanded this functionality so that a localized WebGrid can also be sorted. Previously WebGrid could only be sorted based on the English alphabet, this caused problems when special characters were used in different languages. To overcome this, we have added language, country and variant attributes to the <ig:gridView> tag.

Depending on the values you set, the WebGrid component will be sorted according to that alphabet.



Employee List

First Name ▲	Last Name	Country
Ãba	Davis	GBR
Alexandre	Friesen	USA
Ãmelia	Friesen	USA
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS
Anaya	Okuneva	AUS

Related Topic

Sorting a Localized WebGrid (Section 4.7.3.5.10)

3.2 Revision History

This section makes it easy for you to track down all the new features and enhancements that were made to NetAdvantage for JSF components in the following volume release:

- **2008 Volume 2 (Section 3.2.1)**
- **2008 Volume 1 (Section 3.2.2)**
- **2007 Volume 2 (Section 3.2.3)**
- **2007 Volume 1 (Section 3.2.4)**
- **2006 Volume 2 (Section 3.2.5.2)**

3.2.1 What's New in 2008 Volume 2

The NetAdvantage for JSF 2008 Volume 2 release includes a number of powerful new features and functionalities to allow you to take even more advantage of our JSF components.

Click the links to see a list of the features being offered:

- **Full Support for JSF 1.2 (Section 3.2.1.1)**
- **WebGrid 2008.2 (Section 3.2.1.2)**

3.2.1.1 Full Support for JSF 1.2

In the 2008 Volume 2 release, JavaServer™ Faces (JSF) 1.2 is fully supported. The NetAdvantage for JSF components are tested fully to ensure they function within the JSF 1.2 environment.

JSF 1.2 consists of many new features, bug fixes and improvements, such as:

- Unified Expression Language
- Integration with JavaServer Pages Standard Tag Library (JSTL)
- Component View Creation / Content Interweaving
- Use of Multiple RenderKits

For more information on JSF 1.2, see **Java™ Specification Requests 252 (http://jcp.org/en/jsr/detail?id=252)**.

> **Note:** JSF 1.2 works with Servlet API 2.5 and JavaServer Pages (JSP) 2.1. If you are running Apache™ Tomcat, it must be version 6.0 or higher.

Related Topic

NetAdvantage for JSF Components, JSF 1.1 and JSF 1.2 (Section 1.5)

3.2.1.2 WebGrid 2008.2

Explore the new features that were added to the *WebGrid*™ component in the 2008 Volume 2 release by clicking the links below:

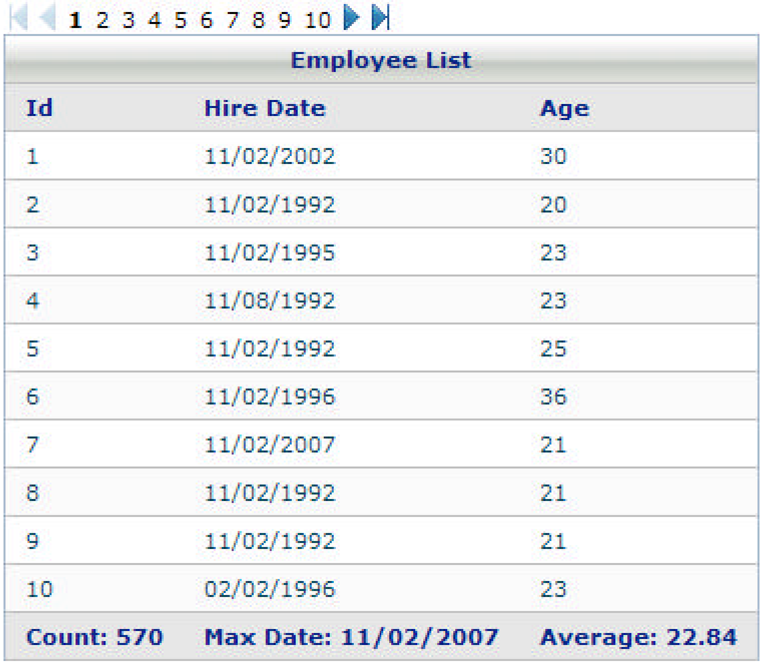
- **Aggregate Functions Support (Section 3.2.1.2.1)**
- **Grid Activation (Section 3.2.1.2.2)**
- **You Can Now Disable Column Moving (Section 3.2.1.2.3)**

3.2.1.2.1 Aggregate Functions Support

Starting in the 2008 Volume 2 release, you can now perform calculations on values in a column and display the results in the column's footer. This feature is extremely useful and beneficial to your end user. For example if your WebGrid has a column that displays monthly sales, displaying the total yearly sales in the column's footer would eliminate the need for your end user to manually calculate the yearly sales themselves.

You can perform the following aggregate functions:

- AVERAGE
- COUNT
- MAXIMUM
- MINIMUM
- SUM



Employee List

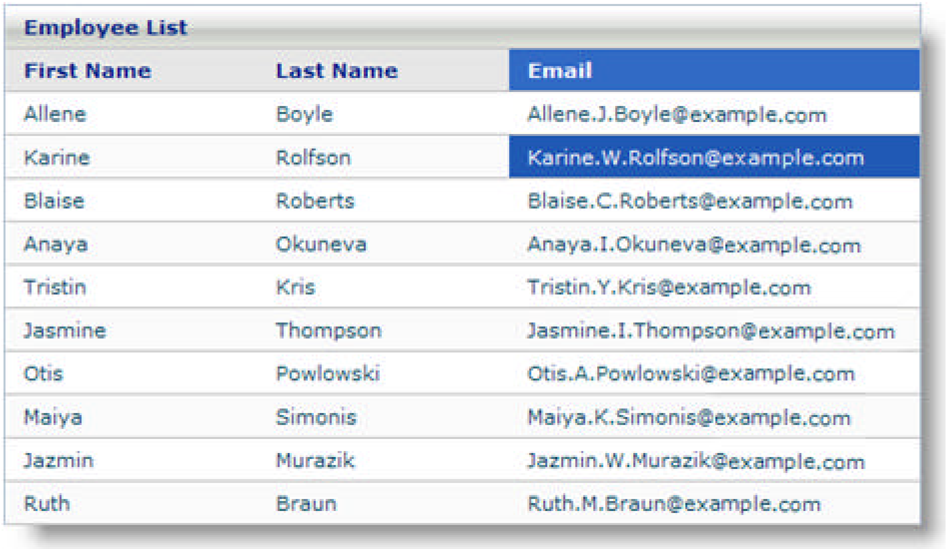
Id	Hire Date	Age
1	11/02/2002	30
2	11/02/1992	20
3	11/02/1995	23
4	11/08/1992	23
5	11/02/1992	25
6	11/02/1996	36
7	11/02/2007	21
8	11/02/1992	21
9	11/02/1992	21
10	02/02/1996	23
Count: 570	**Max Date: 11/02/2007**	**Average: 22.84**

Related Topic

Aggregate Functions (Section 4.7.3.2)

3.2.1.2.2 Grid Activation

Previously, you were only able to navigate through *WebGrid*™ and activate a cell using the mouse. However, we've now added functionality that allows you to navigate and activate cells using both the mouse and the keyboard. To enable this feature you simply add the new **<ig:gridActivation> (Section 4.7.3.5.2)** tag to your WebGrid component.



Employee List

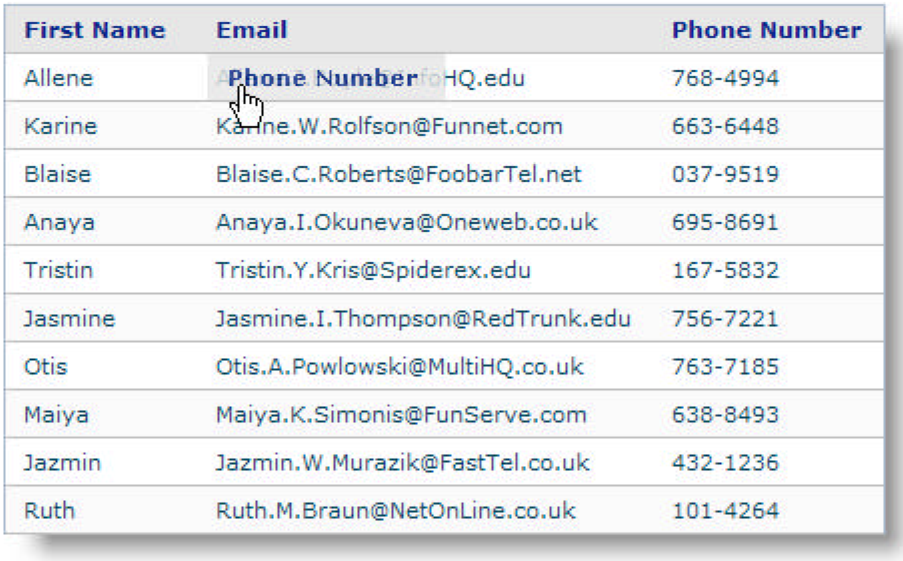
First Name	Last Name	Email
Allene	Boyle	Allene.J.Boyle@example.com
Karine	Rolfson	Karine.W.Rolfson@example.com
Blaise	Roberts	Blaise.C.Roberts@example.com
Anaya	Okuneva	Anaya.I.Okuneva@example.com
Tristin	Kris	Tristin.Y.Kris@example.com
Jasmine	Thompson	Jasmine.I.Thompson@example.com
Otis	Powlowski	Otis.A.Powlowski@example.com
Maiya	Simonis	Maiya.K.Simonis@example.com
Jazmin	Murazik	Jazmin.W.Murazik@example.com
Ruth	Braun	Ruth.M.Braun@example.com

Related Topic

Grid Activation (Section 4.7.3.5.2)

3.2.1.2.3 You Can Now Disable Column Moving

One of the features of the *WebGrid*™ component is that the end user can interact with the WebGrid component by moving the columns to different positions to create a customized appearance that suits their needs. This feature is enabled by default; however, there may be certain situation where you do not want your end users to have the ability to move a column's position. Starting in the NetAdvantage for JSF 2008 Volume 2 release, you can now disable this feature by simply setting the new movableColumns attribute of the <ig:gridView> tag to False. You can also disable column moving on a per column basis by setting the new movable attribute of the <ig:column> or <ig:columnSelectRow> tag to False.



First Name	Email	Phone Number
Allene	A[illegible]HQ.edu	768-4994
Karine	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Jasmine.I.Thompson@RedTrunk.edu	756-7221
Otis	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
Maiya	Maiya.K.Simonis@FunServe.com	638-8493
Jazmin	Jazmin.W.Murazik@FastTel.co.uk	432-1236
Ruth	Ruth.M.Braun@NetOnLine.co.uk	101-4264

Related Topic

Column Moving (Section 4.7.3.3.1)

3.2.2 What's New in 2008 Volume 1

The NetAdvantage for JSF 2008 Volume 1 release includes a number of powerful new features and functionalities to allow you to take even more advantage of our JSF components.

Click the links to see a list of the features being offered:

- **AJAX is Now Supported Within a Portal Environment (Section 3.2.2.1)**
- **New TreeMap Chart (Section 3.2.2.2)**
- **New WebDialogWindowComponent (Section 3.2.2.3)**
- **WebGrid 2008.1 (Section 3.2.2.4)**

3.2.2.1 AJAX is Now Supported Within Portal Environments

In the NetAdvantage for JSF 2008 volume 1 release, the NetAdvantage for JSF components can perform AJAX operations within portal environments. To enable this feature, you just need to make a simple change to the configuration file.

The following portals are supported:

- BEA WebLogic™ 10
- IBM WebSphere® Portal 6.1
- JBoss® 2.4.* and 2.6.* portals

Related Topics

JBoss Portal (Section 1.4.1)

WebLogic Portal (Section 1.4.2)

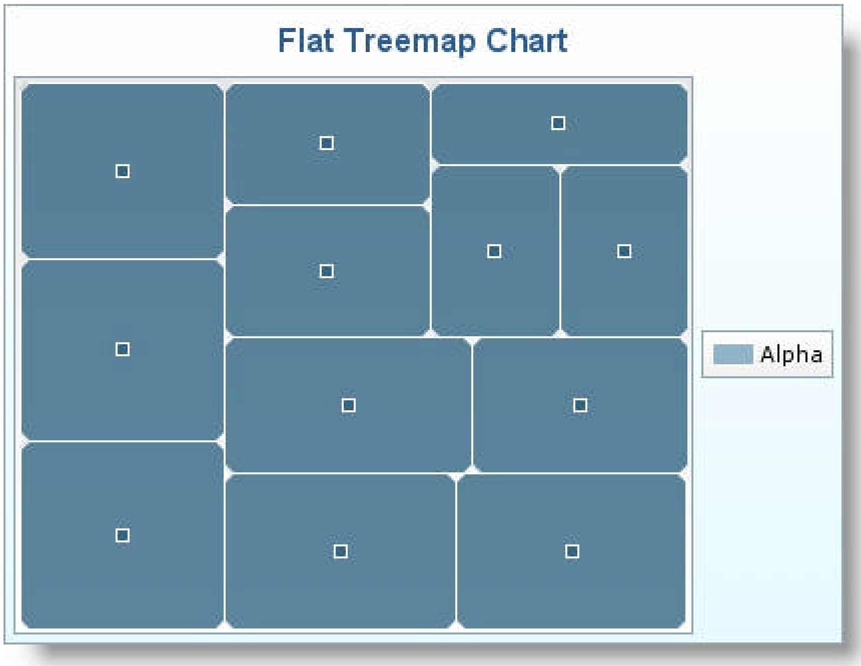
WebSphere Portal (Section 1.4.3)

3.2.2.2 New Treemap Chart

Starting in the NetAdvantage for JSF 2008 Volume 1 release, a new treemap chart has been added to the *WebChart*™ component.

A treemap chart can be used to represent a very large amount of data in a rectangular space. Treemap charts can also be used for monitoring activities with a large quantity of data.

The following screen shot shows a treemap chart.



Related Topic

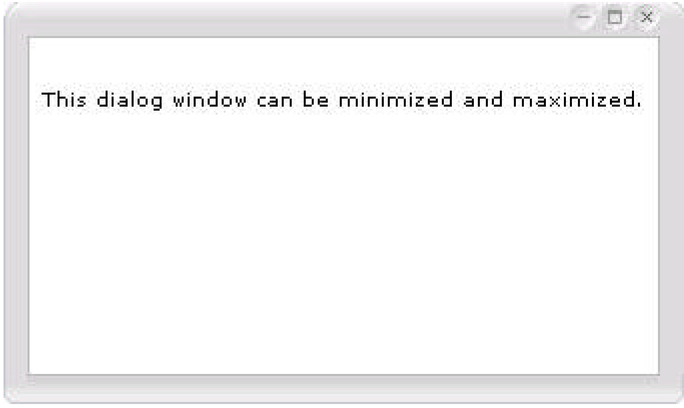
Treemap Chart (Section 4.5.2.12)

3.2.2.3 New WebDialogWindow Component

The WebDialogWindow™ component allows you to add dialog box functionality to your application without having to open a new browser window. The advantage to this is that it is able to bypass many of the pop-up blockers in present-day browsers, while still appearing to the end users as if it were a pop-up window. The dialog window's content area can contain any HTML markup or anything else your application requires. WebDialogWindow supports being displayed as either a modal (the user must interact with and close the dialog box; interaction with the rest of your page is suspended) or modeless (non-modal; the user is free to interact with the rest of your page) dialog box.
A few of the other key features of this JSF dialog control include:

- **Modes** - The WebDialogWindow allows you to set whether the dialog behaves as a modal or modeless (non-modal) dialog would allowing you to better control the flow of your application.
- **Header** - The WebDialogWindow displays a customizable header area into which you can place caption text, several buttons including such stand-by's as the close, minimize and maximize buttons seen in desktop applications.
- **Resizing** - Just like a resizable dialog on the desktop, you can enable your end user to resize the WebDialogWindow at run-time.
- **Location** - Cause the WebDialogWindow to display at the starting location in the page where you want it to appear.
- **Window State** - WebDialogWindow takes care of remembering whether the user has minimized, restored or maximized its appearance.

The following screen shot displays the WebDialogWindow.



This dialog window can be minimized and maximized.

Related Topic

WebDialogWindow (Section 4.6)

3.2.2.4 WebGrid 2008.1

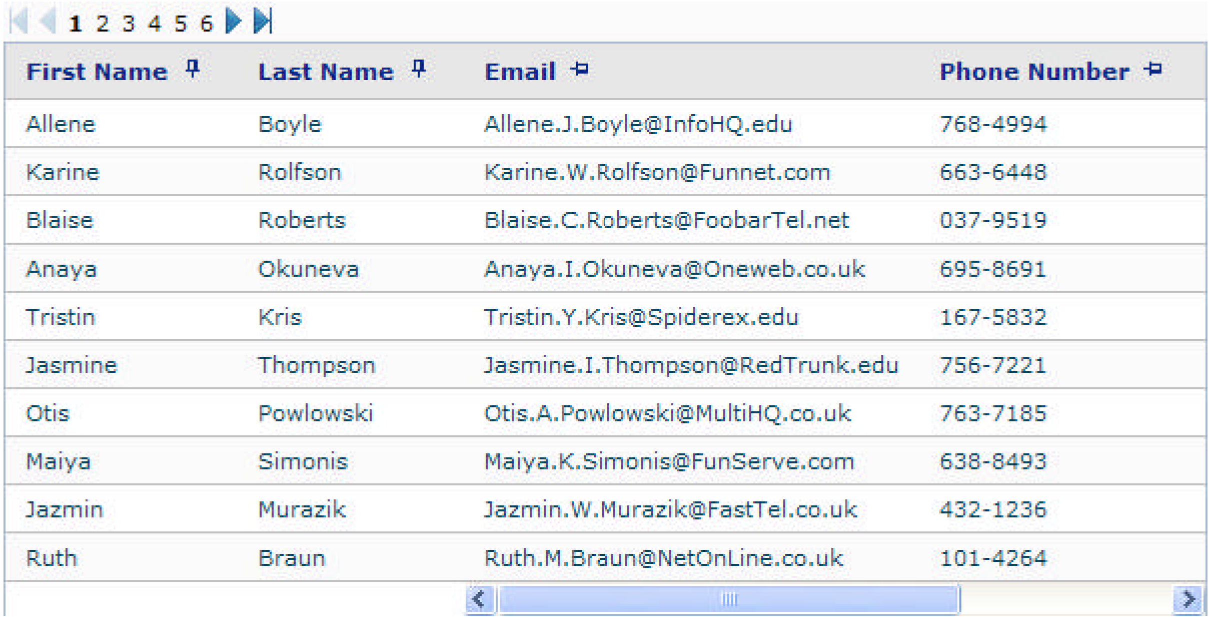
Explore the new features that were added to the *WebGrid*™ component in the 2008 Volume 1 release by clicking the links below:

- **Improved Functionality for Fixed Columns (Section 3.2.2.4.1)**
- **WebGrid Allows Load-On-Demand Scrolling (Section 3.2.2.4.2)**
- **WebGrid Supports Multi-Column Sorting (Section 3.2.2.4.3)**
- **You Can Edit Cells in Your WebGrid (Section 3.2.2.4.4)**

3.2.2.4.1 Improved Functionality for Fixed Columns

In the NetAdvantage for JSF 2008 Volume 1 release we have improved the *WebGrid*'s™ fixed column functionality.

A column indicator is displayed in the header of your WebGrid, next to the column name. When your end user selects this indicator, that column will automatically move to the left of your WebGrid and remain fixed. This column will then remain in view when the grid is scrolled horizontally.



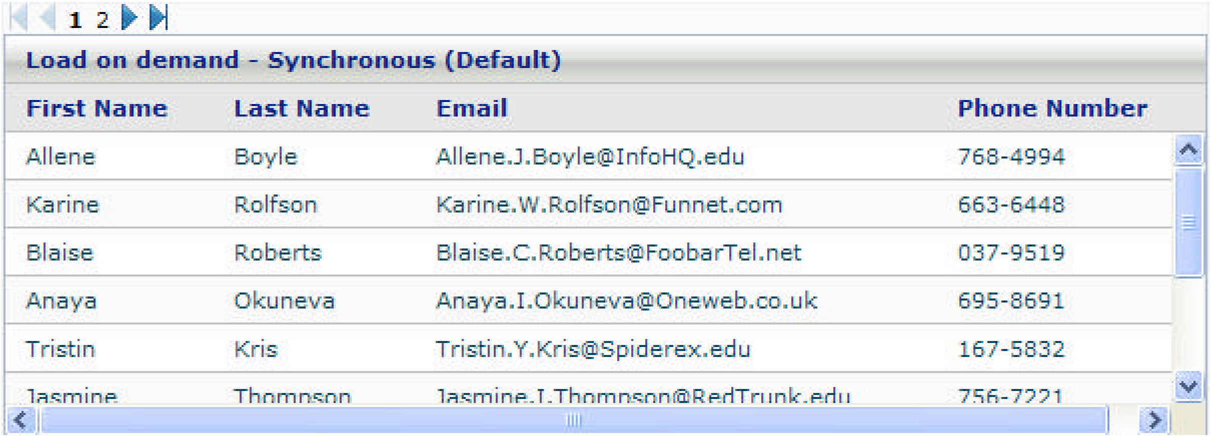
First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221
Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493
Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236
Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264

Related Topic

Fixed Columns (Section 4.7.3.3.2)

3.2.2.4.2 WebGrid Allows Load-On-Demand Scrolling

In the NetAdvantage for JSF 2008 volume 1 release, the WebGrid component now allows load-on-demand scrolling. You can specify a certain number of rows to be rendered by the client. When the end user scrolls through the WebGrid component, WebGrid makes an intelligent call-back to the server to display the specified number of rows. The scroll bar represents the total number of rows in the data set.



Load on demand - Synchronous (Default)

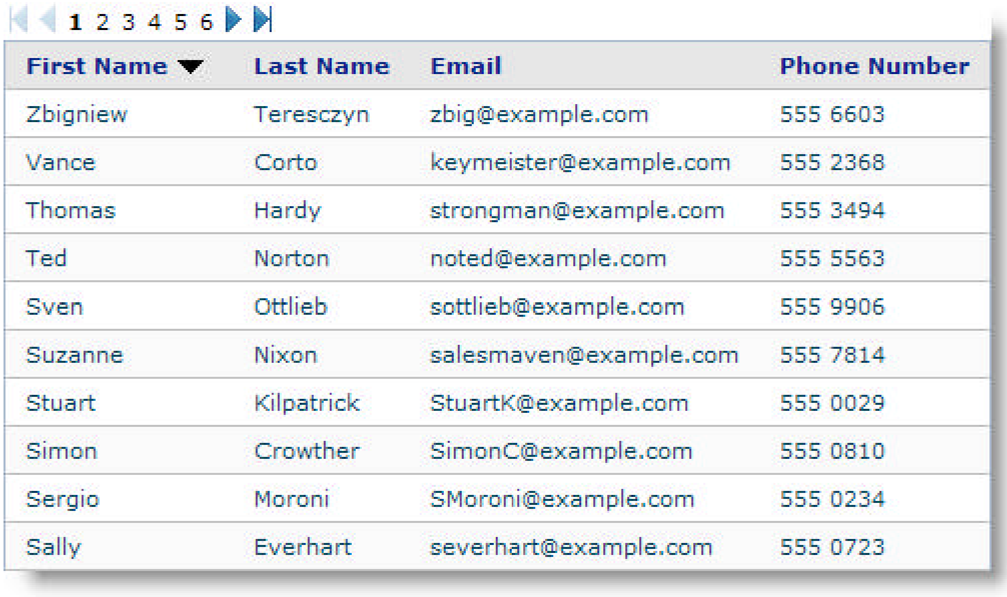
First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221

Related Topic

Load On Demand Scrolling (Section 4.7.3.5.4)

3.2.2.4.3 WebGrid Supports Multi-Column Sorting

You can now configure your WebGrid to allow multi-column sorting. Your end user can apply sort conditions to many columns.



First Name ▼	Last Name	Email	Phone Number
Zbigniew	Teresczyn	zbig@example.com	555 6603
Vance	Corto	keymeister@example.com	555 2368
Thomas	Hardy	strongman@example.com	555 3494
Ted	Norton	noted@example.com	555 5563
Sven	Ottlieb	sottlieb@example.com	555 9906
Suzanne	Nixon	salesmaven@example.com	555 7814
Stuart	Kilpatrick	StuartK@example.com	555 0029
Simon	Crowther	SimonC@example.com	555 0810
Sergio	Moroni	SMoroni@example.com	555 0234
Sally	Everhart	severhart@example.com	555 0723

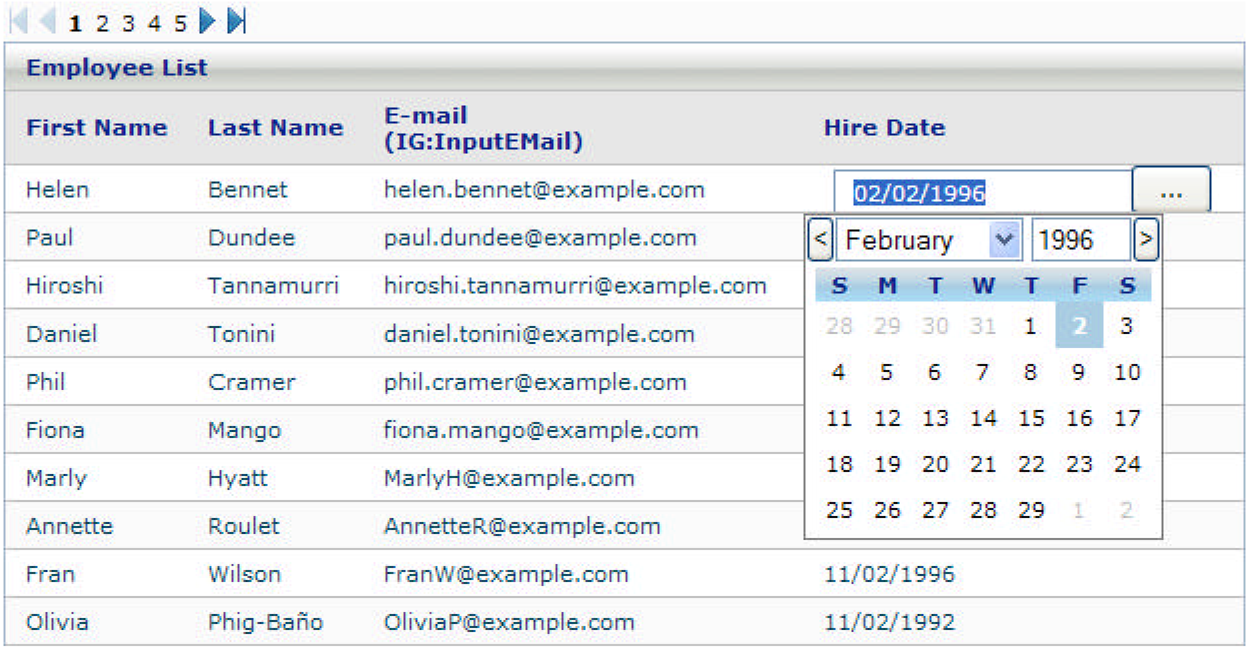
Related Topic

Multi-Column Sorting (Section 4.7.3.5.5)

3.2.2.4.4 You Can Edit Cells in Your WebGrid

In the NetAdvantage for JSF 2008 volume 1 release you can now edit cells in your *WebGrid*. The WebGrid component can switch from display to edit mode. You can specify any valid JSF component to edit the cell value.

The following screen shot shows how a cell can be edited using the Calendar component.



Employee List

First Name	Last Name	E-mail (IG:InputEMail)	Hire Date
Helen	Bennet	helen.bennet@example.com	02/02/1996 ...
Paul	Dundee	paul.dundee@example.com	
Hiroshi	Tannamurri	hiroshi.tannamurri@example.com	
Daniel	Tonini	daniel.tonini@example.com	
Phil	Cramer	phil.cramer@example.com	
Fiona	Mango	fiona.mango@example.com	
Marly	Hyatt	MarlyH@example.com	
Annette	Roulet	AnnetteR@example.com	
Fran	Wilson	FranW@example.com	11/02/1996
Olivia	Phig-Baño	OliviaP@example.com	11/02/1992

< February 1996 >

S	M	T	W	T	F	S
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	1	2

Related Topic

Cell Editing (Section 4.7.3.4.1)

3.2.3 What's New in 2007 Volume 2

The NetAdvantage for JSF 2007 Volume 2 release includes a number of powerful new features and functionalities to allow you to take even more advantage of our JSF components.

Click the links to see a list of the features being offered:

- **New Support for IBM RAD and NetBeans IDEs (Section 3.2.3.1)**
- **WebGrid (Section 3.2.3.2)**

3.2.3.1 New Support for IBM RAD and NetBeans IDEs

Starting in the 2007 Volume 2 release, NetAdvantage for JSF components are now integrated into the available JSP page designer tools palette for IBM® Rational® Application Developer for WebSphere® 7.0 and NetBeans™ 5.5. This new IDE support helps you to create applications using NetAdvantage for JSF components easier and quicker than ever before.

Design View

With the support for integration of the NetAdvantage for JSF components into RAD and NetBeans IDEs, you can take full advantage of the tools palette available within these IDEs. You can drag the components from the palette to your JSP page, and control the layout and size of the components.

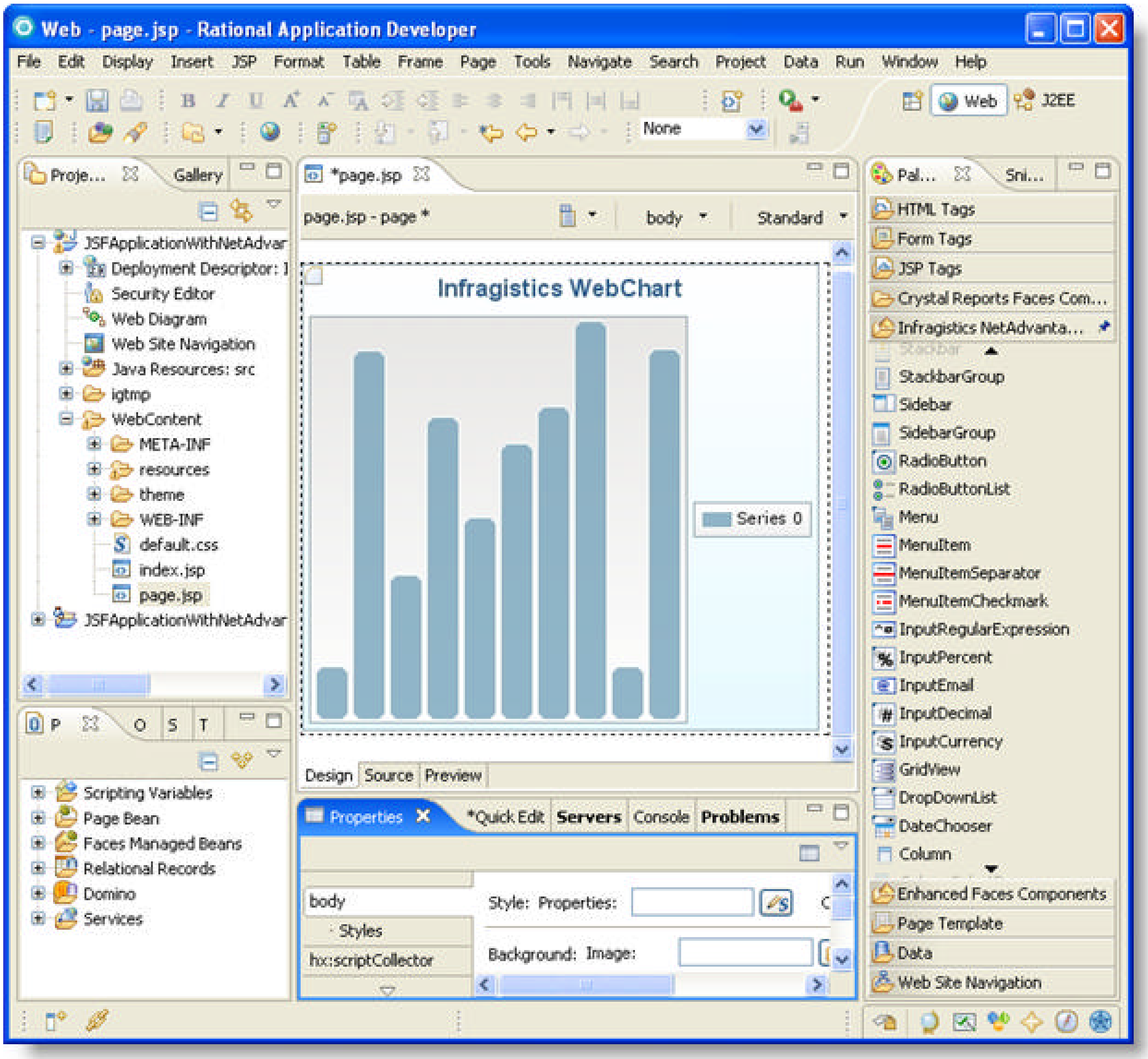
Properties Window

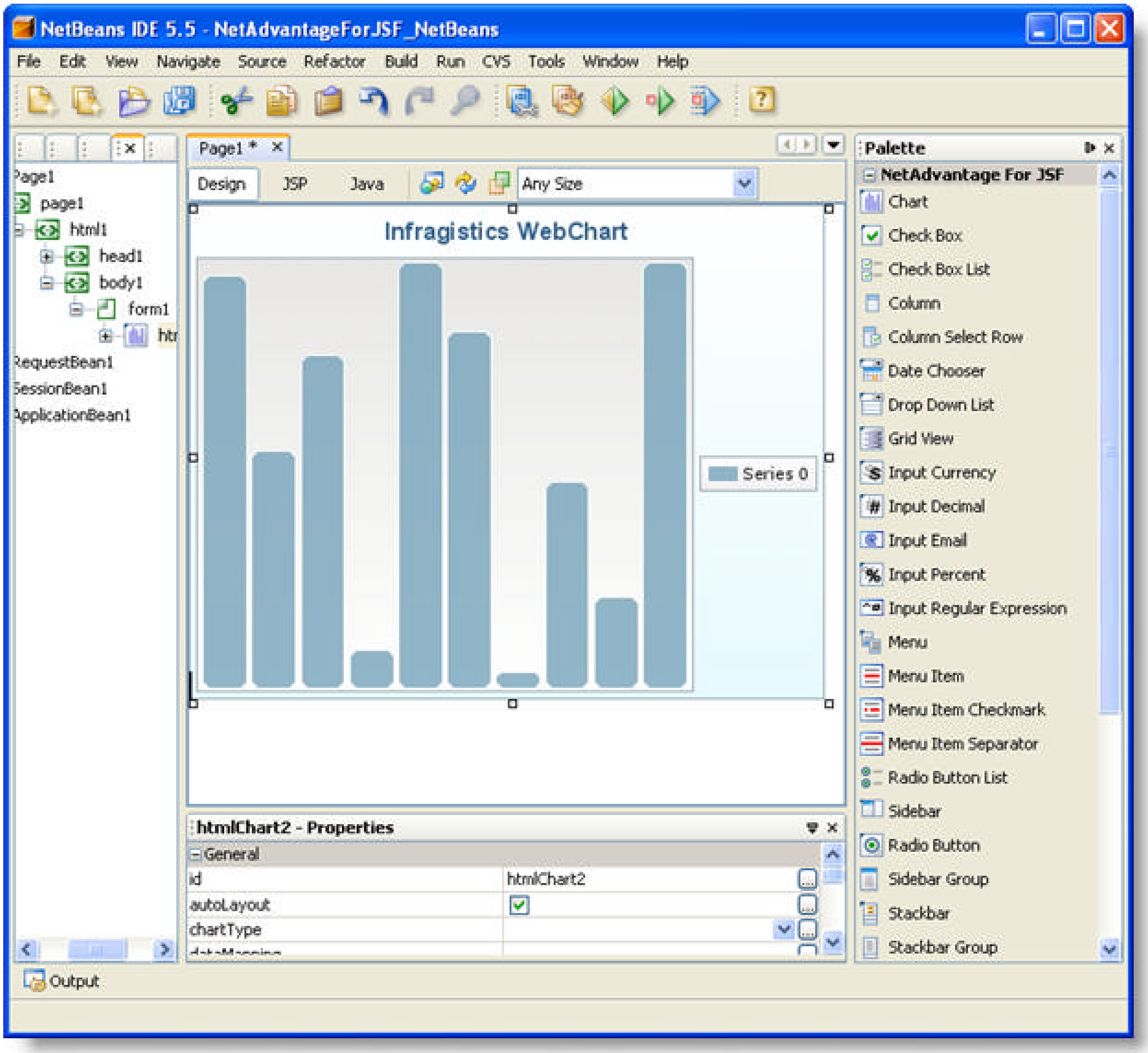
Using the Properties window, you can edit the properties of your component and instantly see the changes in the design view, simplifying the editing process. Selecting a component from the design view populates the Properties window with the attributes and values of that component.

Preview

Additionally, you can also preview your application within the design view eliminating the need to compile your application to view it.

The following screen shots show the NetAdvantage for JSF components integrated within RAD and NetBeans.


Web - page.jsp - Rational Application Developer
File Edit Display Insert JSP Format Table Frame Page Tools Navigate Search Project Data Run Window Help
None
Web
J2EE
Proje... Gallery
JSFApplicationWithNetAdvar
Deployment Descriptor: J
Security Editor
Web Diagram
Web Site Navigation
Java Resources: src
igtmp
WebContent
META-INF
resources
theme
WEB-INF
default.css
index.jsp
page.jsp
JSFApplicationWithNetAdvar
P O S T
Scripting Variables
Page Bean
Faces Managed Beans
Relational Records
Domino
Services
*page.jsp
page.jsp - page *
body
Standard
Infragistics WebChart
Series 0
Design Source Preview
Properties
*Quick Edit Servers Console Problems
body
Styles
hx:scriptCollector
Style: Properties:
Background: Image:
Pal... Sni...
HTML Tags
Form Tags
JSP Tags
Crystal Reports Faces Com...
Infragistics NetAdvanta...
Stackbar
StackbarGroup
Sidebar
SidebarGroup
RadioButton
RadioButtonList
Menu
MenuItem
MenuItemSeparator
MenuItemCheckmark
InputRegularExpression
InputPercent
InputEmail
InputDecimal
InputCurrency
GridView
DropDownList
DateChooser
Column
Enhanced Faces Components
Page Template
Data
Web Site Navigation



htmlChart2 - Properties

General

id	htmlChart2
autoLayout	☑
chartType	

Related Topic

IDE Support (Section 1.3)

3.2.3.2 WebGrid 2007.2

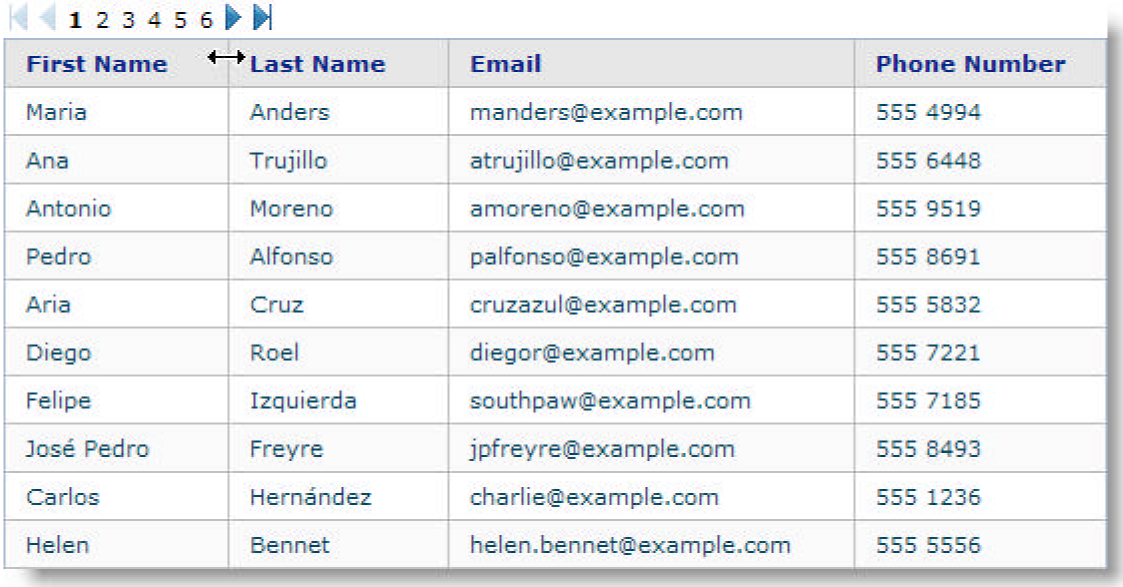
Explore the new features that were added to the *WebGrid*™ component in the 2007 Volume 2 release by clicking the links below:

- **WebGrid Columns Can Be Resized (Section 3.2.3.2.1)**
- **WebGrid Now Supports Vertical Scroll Bars (Section 3.2.3.2.2)**

3.2.3.2.1 WebGrid Columns Can Be Resized

In the 2007 Volume 2 release, you can now allow columns to be resized. With this new feature the end user can shrink or expand the columns of a grid, allowing them to have more control over the grid layout.

The following screen shot shows a grid with the Resizing Columns feature enabled.



First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

Related Topic

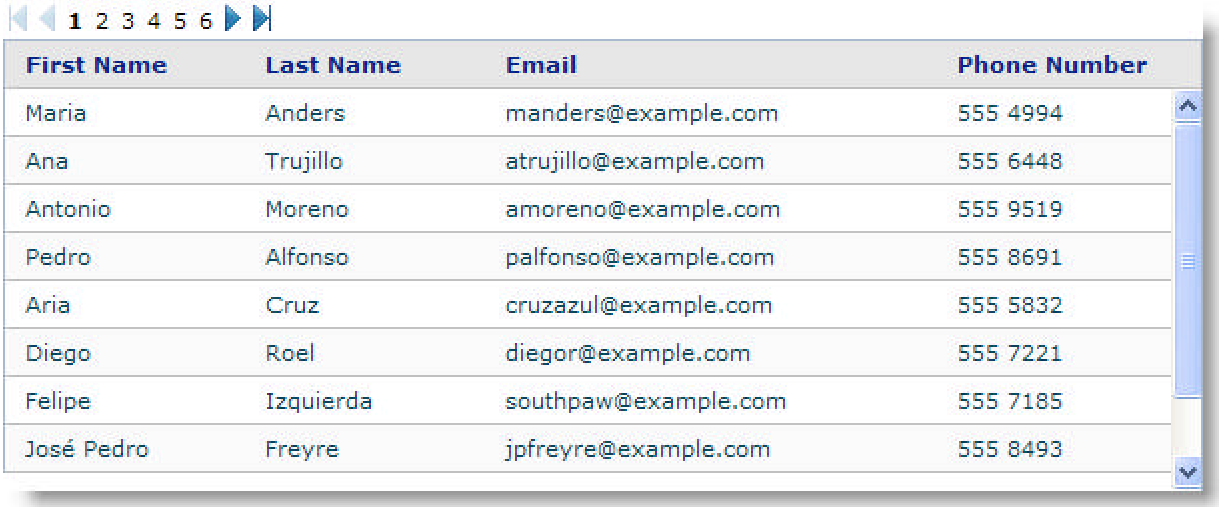
Resizable Columns (Section 4.7.3.3.3)

3.2.3.2.2 WebGrid Now Supports Vertical Scroll Bars

Now in the 2007 Volume 2 release, not only can you apply horizontal scroll bars to your grid, but you can also apply vertical scroll bars. With both horizontal and vertical scroll bars, large amounts of data can be displayed in a tight and tidy grid.

When the contents of your grid exceeds the specified height, a vertical scroll bar is automatically displayed, allowing your end users to scroll through the data. While scrolling through the grid, the header and footer remain stationary and do not scroll out of view.

The following screen shot shows a grid with the Vertical Scroll Bars feature enabled.



First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493

Related Topic

Vertical Scroll Bars (Section 4.7.3.3.6)

3.2.4 What's New in 2007 Volume 1

The NetAdvantage for JSF 2007 Volume 1 release includes a number of powerful new features and functionalities to allow you to take even more advantage of our JSF components.

Below is a list of the components that we added for the 2007 Volume 1 release. Click the links to see a list of the features being offered.

- **WebChart (Section 4.5)**
- **WebGrid (Section 3.2.4.2)**

3.2.4.1 WebChart 2007.1

Explore the new *WebChart*™ control that was added to our suite of controls and components in the 2007 Volume 1 release by clicking the link below.

- **New WebChart Component (Section 3.2.4.1.1)**

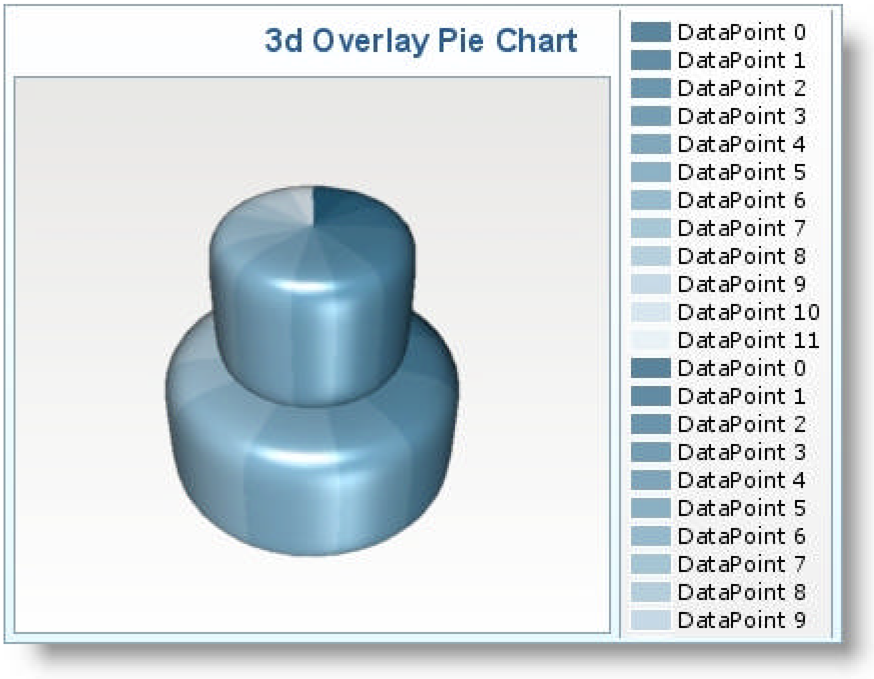
3.2.4.1.1 New WebChart Component

WebChart™ is a new and powerful component that has been added to the 2007 Volume 1 release. The WebChart component allows you to display information visually in the form of tables or graphs.

WebChart allows you to easily create different charts such as pie, doughnut, column and bar. Also specialized charts such as OHLC and candlestick charts can be created. The charts can be displayed in 2D or 3D mode and can also be grouped into different groupings such as stack, stack100, overlay or cluster.

The WebChart component allows you to combine different charts and series together to let the user quickly see the data represented on different charts at once.

ToolTips can be implemented to allow the user interact with the WebChart. Legends, labels and axes can be customized for WebChart.



Related Topic

WebChart (Section 4.5)

3.2.4.2 WebGrid 2007.1

Explore the powerful new *WebGrid*™ feature that was added in the 2007 Volume 1 release:

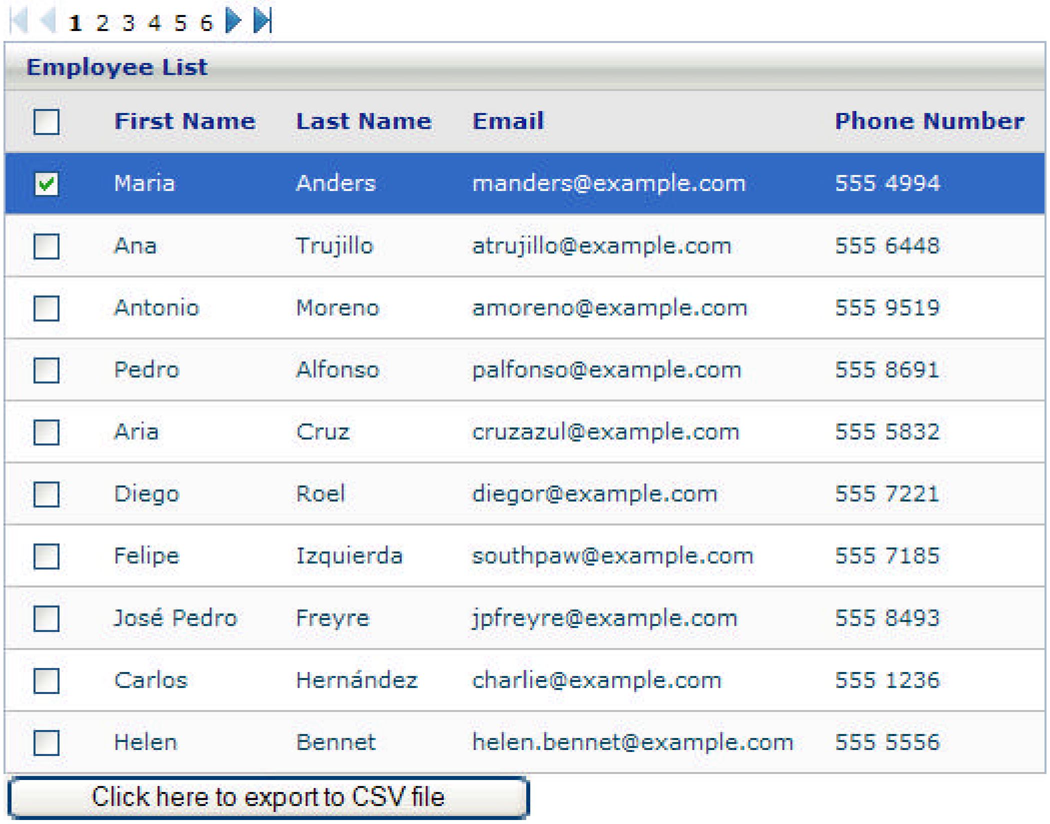
- **Data from WebGrid Can Now Be Exported to a CSV File (Section 3.2.4.2.1)**

3.2.4.2.1 Data from WebGrid Can Now Be Exported to a CSV File

In the 2007 Volume 1 release, we enhanced the *WebGrid*™ components, so that you can enable your end users to export data to a Comma Separated Value (CSV) file.

The export method allows the following type of data to be exported to CSV file:

- data on the current page
- data in the selected rows



Employee List

□	First Name	Last Name	Email	Phone Number
☑	Maria	Anders	manders@example.com	555 4994
□	Ana	Trujillo	atrujillo@example.com	555 6448
□	Antonio	Moreno	amoreno@example.com	555 9519
□	Pedro	Alfonso	palfonso@example.com	555 8691
□	Aria	Cruz	cruzazul@example.com	555 5832
□	Diego	Roel	diegor@example.com	555 7221
□	Felipe	Izquierda	southpaw@example.com	555 7185
□	José Pedro	Freyre	jpfreyre@example.com	555 8493
□	Carlos	Hernández	charlie@example.com	555 1236
□	Helen	Bennet	helen.bennet@example.com	555 5556

Related Topic

Exporting Data to a CSV File (Section 4.7.3.1.4)

3.2.5 What's New in 2006 Volume 2

The NetAdvantage for JSF 2006 Volume 2 release includes a number of powerful new features and functionalities to allow you to take even more advantage of our NetAdvantage for JSF components.

Click the links below to gain access to information describing the new features being offered in the 2006 Volume 2 release:

- **Facelet and Portal Support (Section 3.2.5.1)**
- **WebGrid (Section 3.2.5.2)**

3.2.5.1 Facelet and Portal Support 2006.2

We are proud to offer new support for facelets and portals in the 2006 Volume 2 release. Click the link below for details.

- **JSF Components Now Support Facelets and Portals (Section 3.2.5.1.1)**

3.2.5.1.1 JSF Components Now Support Facelets and Portals

In response to various requests from customers for facelet and portal support, we have enhanced the NetAdvantage for JSF product so that our components can now work in facelets and portals.

Facelet Support

Starting in the 2006 Volume 2 release, the NetAdvantage for JSF components can be used within facelets. Facelets provides a highly performant, JSF-centric view technology that can be used in place of JSP. Anyone who has created a JSP page will be able to do the same with Facelets. Facelets provides a light-weight templating framework that lets you create JSF views easily using HTML-style templates. Facelets can be run on any Java EE application server with the addition of a few JAR files.

To use our components in a facelets container, you need the following tag library: netAdvantage.facelet.taglib.xml. This tag library is located in the infragistics-netadvantage.jar file.

Portal Support

In addition to support for facelets, NetAdvantage for JSF has added support for portlets. Portlets are web components--like servlets--specifically designed to be aggregated in the context of a composite page.

Starting in 2006 Volume 2, you can use the full suite of NetAdvantage for JSF components in a portlet that runs on a portal server. This support allows you to create JSP pages with our components, and run the application as a portlet on a portal server.

The following is a list of the portal servers that NetAdvantage for JSF supports:

- JBoss Portal
- WebLogic
- WebSphere

Related Topics

Facelet Support (Section 1.7.6)

Portlet Support (Section 1.4)

3.2.5.2 WebGrid 2006.2

Explore the new features of the *WebGrid*™ components that were added in the 2006 Volume 2 release:

- **Columns and Headers Can Now Be Fixed (Section 3.2.5.2.1)**
- **Display Grid Data in a Hierarchical View (Section 3.2.5.2.2)**
- **New Event Triggered When Row is Selected (Section 3.2.5.2.3)**

3.2.5.2.1 Columns and Headers Can Now Be Fixed

Starting in the 2006 Volume 2 release, the *WebGrid*™ components now have the ability to lock a specified number of the left-most columns. This allows the grid to be split into two sections. These left-most columns will be locked and won't scroll, allowing end users to horizontally scroll the columns on the right-hand side so they can keep the important data in the grid visible at all times.

The number of columns that are fixed is controlled by the fixedColumnCount parameter of the grid. The GridView class also has a setter and getter for this parameter.

Related Topic

Fixed Columns (Section 4.7.3.3.2)

3.2.5.2.2 Display Grid Data in a Hierarchical View

Starting in the 2006 Volume 2 release, you can now display the data of your grid in a hierarchical view so that the grid is nested inside the rows of the grid. With this feature, end users can easily navigate your hierarchical data by expanding and collapsing the rows.

As part of this Hierarchical Grid feature, a new facet tag was added to the grid called masterDetail. The contents of this facet is a template of the nested grid. Adding nested grids involves adding this new facet to the column where you want the nested grid to appear. In most cases, this will be the first column, but there is no restriction.

The following API changes have also been made to support this new feature:

- Column.getMasterDetail() -- Getter used to return masterDetail facet defined for columns.
- Column.setMasterDetail(UIComponent masterDetail) -- Setter used to create a masterDetail facet defined for columns.
- Row.isExpanded() -- Returns a boolean, specifying whether a row is expanded or collapsed.
- Row.setExpanded(boolean expanded) -- Programmatically expands or collapses a row.

Related Topics

Bind a Hierarchical WebGrid to Data (Section 4.7.3.1.2)

3.2.5.2.3 New Event Triggered When Row is Selected

Starting in 2006 Volume 2, you can take advantage of a new event that has been added to the *WebGrid™* components that can be triggered when the end user selects/un-selects a row selection checkbox in the grid.

To allow for the row selection event to be triggered, we've enhanced the WebGrid components as follows:

- Added the SelectedRowsChangeEvent and SelectedRowsChangeListener classes
- Added the following methods to the grid class to get/set a row selection listener:
  - grid. getSelectedRowsChangeListener()
  - grid. setSelectedRowsChangeListener (MethodBinding md);
- Added a method to retrieve the list of all registered RowSelectionListeners:
  - SelectedRowsChangeListener[] grid. getSelectedRowsChangeListeners()
- Added a tag to the grid component. This tag is used to specify the method is called when row selection checkbox is selected or un-selected.
  - selectedRowsChangeListener

Related Topic

Row Selection (Section 4.7.3.5.7)

4 Components

This section contains information about the NetAdvantage for JSF components. This information provides you with a better understanding of the capabilities of the components.

- **Accessibility Overview (Section 4.1)**
- **AJAX Functionality using our Smart-Refresh Technology (on-line documentation)**
- **Binding JSF Components to Data (Section 4.2)**
- **Getting Started with the Client Side Object Model (Section 4.3)**
- **WebBar (Section 4.4)**
- **WebChart (Section 4.5)**
- **WebDialogWindow (Section 4.6)**
- **WebGrid (Section 4.7)**
- **WebInput (Section 4.8)**
- **WebMenu (Section 4.9)**
- **WebTab (Section 4.10)**
- **WebTree (Section 4.11)**

4.1 Accessibility Overview

Section 508 (http://www.section508.gov/) of the Rehabilitation Act was amended in 1998 by Congress to require all Federal agencies to make their electronic and information technology accessible to people with disabilities. Since then, Section 508 compliance has not only been a requirement in government agencies, but it's also important when providing software solutions and designing Web pages.

Section 1194.21 of the Section 508 law specifically targets software applications and operating systems, and contain a set of 16 rules to follow. In order to enable you to keep your Web applications and Web sites compatible with these rules with minimal effort on your part, we've ensured that our JSF components are compliant with the accessibility rules.

The matrix below provides a high-level outline of the accessibility support provided by our visual controls (and related components). To learn more about an individual component's accessibility compliance, click the name of the component. You can also visit the **Section 508 page (http://www.infragistics.com/learn/accessibility.aspx)** on our Web site for more information on our commitment to accessibility.

Component	Section 508 Compliance											
	(a)	*(b)*	*(c)*	*(d)*	*(e)*	*(f)*	*(g)*	*(h)*	*(i)*	*(j)*	*(k)*	*(l)*
WebBar (Section 4.4.3)												
WebChart (Section 4.5.5)												
WebDialogWindow (Section 4.6.4)												
WebGrid (Section 4.7.4)												
WebInput (Section 4.8.2)												
WebMenu (Section 4.9.4)												
WebTab (Section 4.10.3)												
WebTree (Section 4.11.2)												

4.1.1 Legend

- ✓ -- The component is completely accessible in this particular area.
- 'white space' -- this particular rule does not apply to the component.
- ✗ -- The component is not entirely accessible unless you perform some sort of action.

4.2 Binding JSF Components to Data

To display data using JSF components, you can bind the components' tag attribute values, or the components themselves, to a data source.

The way in which you bind the NetAdvantage for JSF components to data will depend on the component type, as well as how you want the data to be displayed/updated:

- **Value Binding** -- With this type of binding, you bind the value of a component's tag attribute to the property of a managed bean using expressions of the JSF expression language (EL). Value binding is useful when you want to bind attributes or properties to dynamically calculated results. You may also want to use value binding to bind a component to the managed bean; this would allow you to programmatically access the component in the managed bean in the future.
- **Method Binding** -- With this type of binding, you bind specific tag attributes (e.g., listeners) of a component to a method of a managed bean using expressions of the JSF EL. Unlike value binding which is useful for dynamic retrieval and setting of properties, method binding is useful when you want an arbitrary method (that handles navigation, validation, or event handling) to be called.

Note: For details on JSF component binding, please refer to chapter 5 of Sun's JavaServer Faces Specification, version 1.1, which is available for download at **Sun's JavaServer Faces Technology Download (http://java.sun.com/javaee/javaserverfaces/download.html)** Web site.

The sample code below demonstrates how you can bind the WebTree component to data using dataSource property binding.

In JSP:

```
<ig:treeView
    dataSource="#{shared_companyDAO.companies}"
    pagesize="10"
    style="background-color: #f0f0f0; border: 1px solid silver;
           height: 200; width: 300">
    <%-- one node for each company --%>
    <ig:treeNode value="#{DATA_ROW.companyName}">
       <%-- node "customers" --%>
       <ig:treeNode dataSource="#{DATA_ROW.customers}" value="customers">
          <%-- one node for each customer --%>
          <ig:treeNode value="#{DATA_ROW.customerName}" />
       </ig:treeNode>
       <%-- node "employees" --%>
       <ig:treeNode dataSource="#{DATA_ROW.employees}" value="employees">
          <%-- one node for each employee --%>
          <ig:treeNode value="#{DATA_ROW.employeeName}" />
       </ig:treeNode>
    </ig:treeNode>
</ig:treeView>
```

4.3 Getting Started with the Client Side Object Model

A JavaScript-based object model, also known as a Client-Side Object Model (CSOM), is available on the client for each of our NetAdvantage for JSF components. This CSOM lets you build rich, responsive, Web applications which keep post-backs to a minimum.

Each and every component has a new tag in the form <ig:*nnnn*ClientEvents> where *nnnn* is the name of the component and you can assign your event listeners declaratively through these Client Event settings.

The following code demonstrates how to use client events for the WebDateChooser component.

In JSP:

```
<ig:dateChooser>
   <ig:dateChooserClientEvents popupOpening="DateChooserOpening" popupClosing="DateCh
</ig:dateChooser>
```

DateChooserOpening and DateChooserClosing are Javascript functions defined elsewhere on your page.

Usage

The NetAdvantage for JSF CSOM events typically have two event listeners exposed per event. The naming convention is that the events have "ing" and "ed" endings to their names. The event listener with "ing" ending is called before the event is handled by our JSF components, allowing you to perform your own processing or even cancelling the event. The event listeners with "ed" ending is called after the event has been processed by the JSF component.

The following table lists each of the events associated with each NetAdvantage for JSF component.

Component	Event
<ig:barClientEvents>	groupExpanding groupExpanded groupCollapsing groupCollapsed
<ig:chartClientEvents>	dataPointClicked
<ig:checkboxClientEvents>	valueChanging valueChanged onFocus onBlur
<ig:checkboxClientEvents>	selectionChanging selectionChanged onFocus onBlur
<ig:dateChooserClientEvents>	popupOpening popupOpened onChange onFocus onBlur
<ig:dropDownClientEvents>	autoFilterStarting

	autoFilterStarted buttonMouseDown buttonMouseOut buttonMouseOver valueChanging valueChanged focus blue keyDown selectedIndexChanging selectedIndexChanged dropDownClosing dropDownClosed dropDownOpening dropDownOpened itemsRequested blurItem
<ig:dropDownListClientEvents>	onChange onFocus onBlur
<ig:dwClientEvents>	ajaxRequesting ajaxResponseCompleted
<ig:gridClientEvents>	cellClicked columnSorting columnSorted rowExpanding rowExpanded pageChanging pageChanged rowSelecting rowSelected cellUpdating cellUpdated columnMoving columnMoved columnFixing columnFixed ajaxRequesting ajaxResponseCompleted
<ig:inputCurrencyClientEvents>	onChange onSelect onFocus onBlur

<ig:inputDecimalClientEvents>	onChange onSelect onFocus onBlur
<ig:inputEmailClientEvents>	onChange onSelect onFocus onBlur
<ig:inputPercentClientEvents>	onChange onSelect onFocus onBlur
<ig:inputRegularExpressionClientEvents>	onChange onSelect onFocus onBlur
<ig:menuClientEvents>	itemClicked
<ig:radioButtonClientEvents>	valueChanging valueChanged onFocus onBlur
<ig:radioButtonListClientEvents>	selectedItemChanging selectedItemChanged onFocus onBlur
<ig:tabClientEvents>	selectedTabChanging selectedTabChanged ajaxRequesting ajaxResponseCompleted
<ig:treeClientEvents>	nodeClicked nodeExpanding nodeExpanded nodeCollapsing nodeCollapsed pageChanging pageChanged ajaxRequesting ajaxResponseCompleted

4.4 WebBar

Click the links below to find information specific to the *WebBar*™ components.

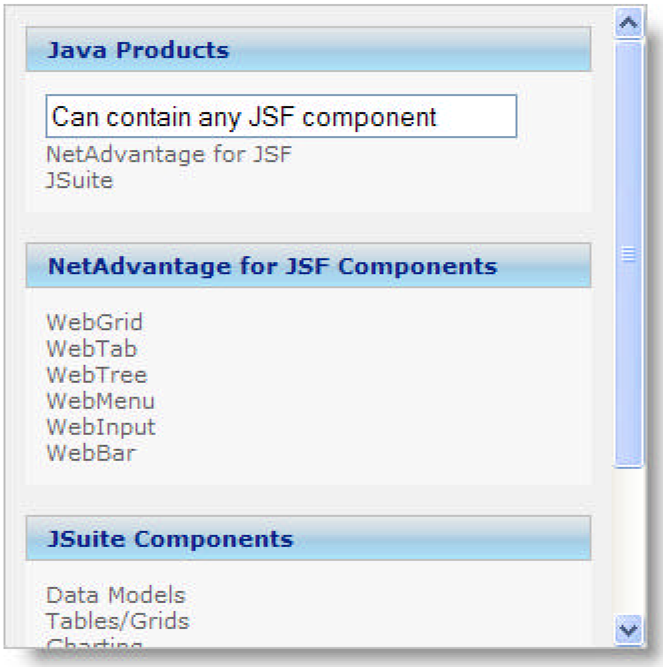
- **About the WebBar Components (Section 4.4.1)** -- Provides information about the key functionalities of the WebBar components.
- **Set the Expanded Icon (Section 4.4.2)** -- Allows you to display the state of WebBar to your end users with the use of icons.
- **WebBar Accessibility Compliance (Section 4.4.3)** -- This topic outlines the way in which WebBar meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebBar control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.4.1 About the WebBar Components

The *WebBar*™ component group consists of the Sidebar and Stackbar components, and are primarily used for for navigation.

The Sidebar component can contain a set of SidebarGroup components, and the Stackbar component can contain a set of StackbarGroup components. The SidebarGroup and StackbarGroup components are simple containers that can contain any type of component. A SidebarGroup or StackbarGroup can be expanded or collapsed by clicking on the group's name.

The following two screenshots show the WebBar's Sidebar and Stackbar components.



Java Products

Can contain any JSF component

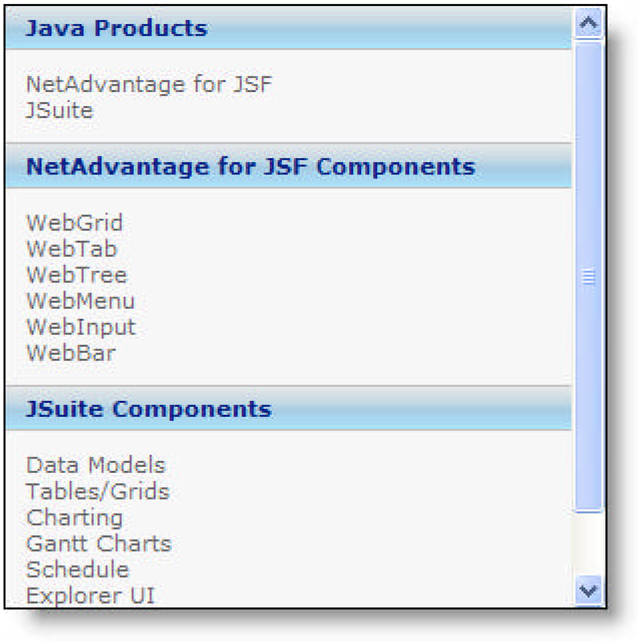
NetAdvantage for JSF
JSuite

NetAdvantage for JSF Components

WebGrid
WebTab
WebTree
WebMenu
WebInput
WebBar

JSuite Components

Data Models
Tables/Grids
Charting



Java Products

NetAdvantage for JSF
JSuite

NetAdvantage for JSF Components

WebGrid
WebTab
WebTree
WebMenu
WebInput
WebBar

JSuite Components

Data Models
Tables/Grids
Charting
Gantt Charts
Schedule
Explorer UI

4.4.2 Set the Expanded Icon

You can set two different icon attributes on your stack bar to display the state of the bar to your end user.

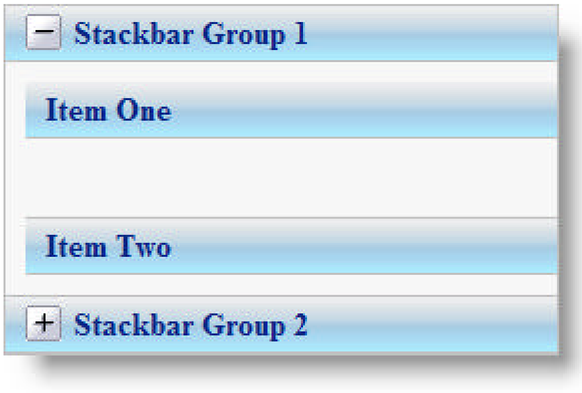
The iconURL attribute specifies the icon to be displayed to your end user when the stack bar is collapsed.

The expandedIconUrl specifies the icon to be displayed to your end user when the stack bar is expanded.

The following code example demonstrates how to set these icons.

In JSP:

```
<ig:stackbarGroup text="Stackbar Group Title"
     iconURL = "/img/collapseIcon"
     expandedIconUrl = "/img/expandedIcon">
```



4.4.3 WebBar Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.5 WebChart

Click the links below to find information specific to the *WebChart™* components.

- **Understanding WebChart (Section 4.5.1)** -- Contains topics that will help you understand the WebChart component.
- **Chart Types (Section 4.5.2)** -- Describes the different chart types available.
- **Group Types (Section 4.5.3)** -- Describes the different group types available.
- **Using WebChart (Section 4.5.4)** -- Describes how to use the different features of WebChart.
- **WebChart Accessibility Compliance (Section 4.5.5)** -- This topic outlines the way in which WebChart meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebChart control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.5.1 Understanding WebChart

This is a great place for you to get an introduction to the *WebChart*™ component, and its key features and functionalities. The topics in this section will give you a better idea of why you would want to use the WebChart component in your applications.

- **About WebChart (Section 4.5.1.1)** -- Introduces you to the WebChart component.
- **WebChart Gallery (Section 4.5.1.2)** -- Displays a gallery of the types of charts that can be created.
- **Feature Overview (Section 4.5.1.3)** -- A list of some of the key features in WebChart

4.5.1.1 About WebChart

Charting is used to visually display information, presenting data in the form of graphs or tables. *WebChart™* is the JSF implementation of the charting component. The WebChart component provides a set of charts and a framework for creating charting applications, or adding charting capability to existing data or information-driven applications.

WebChart allows developers to set up different types of charts such as **Pie (Section 4.5.2.9)**, **Doughnut (Section 4.5.2.6)**, **Column (Section 4.5.2.5)** and **Area (Section 4.5.2.3)** Charts. 2D and 3D modes are available for the most common chart types. For more demanding Web applications, you can combine several charts together, customize the chart legend, and capture user interactions with the chart through its comprehensive Event Model.

Additionally charts can also be grouped into different group types that are available such as **Stack (Section 4.5.3.2)**, **Stack100 (Section 4.5.3.3)**, **Overlay (Section 4.5.3.4)** and **Cluster (Section 4.5.3.1)**.

See the **Webchart Gallery (Section 4.5.1.2)** for a example of the types of charts that can be created.

4.5.1.2 WebChart Gallery

There are many different types of *WebCharts*™ that can be created using different attributes within the <ig:chart> tag.

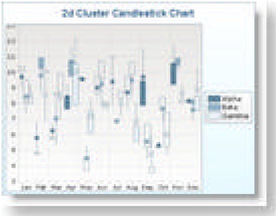
WebCharts can be created using different combinations of chartType and groupType. They can be 2D or 3D, or they can be composite charts.

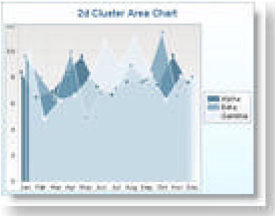
Click on the following links to see the more popular 2D or 3D WebChart combinations.

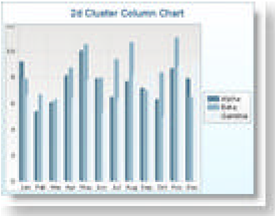
- **WebChart 2D Gallery (Section 4.5.1.2.1)**
- **WebChart 3D Gallery (Section 4.5.1.2.2)**

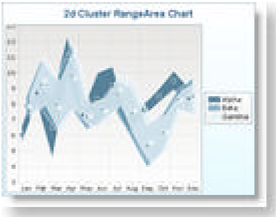
4.5.1.2.1 WebChart 2D Gallery

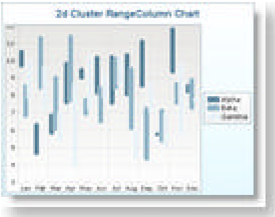
The screen shots below are some of the more popular 2D WebChart combinations.

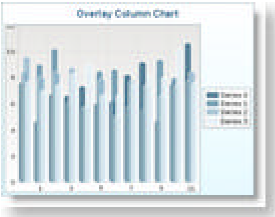


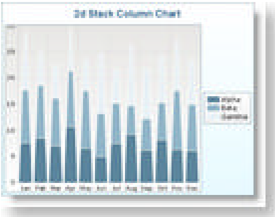


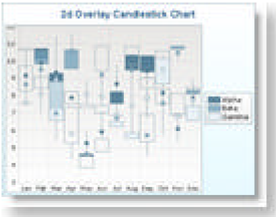


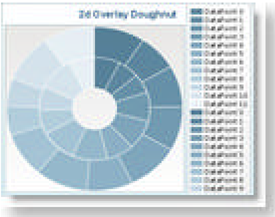


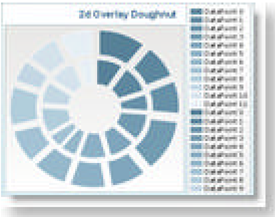


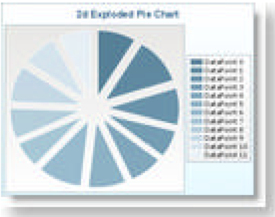


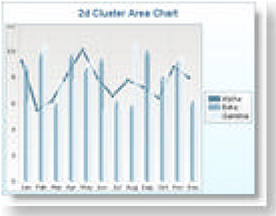


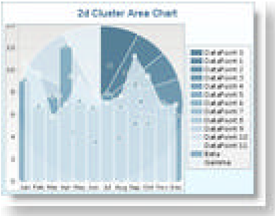


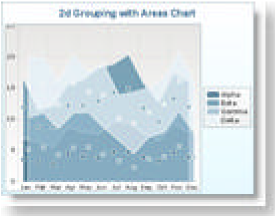






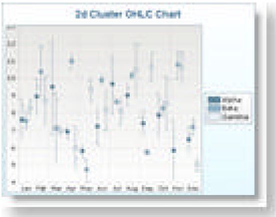


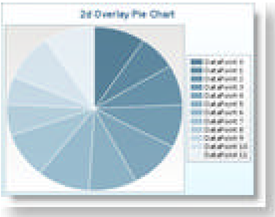


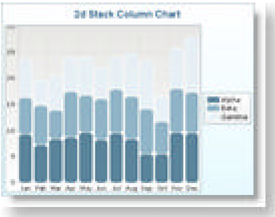


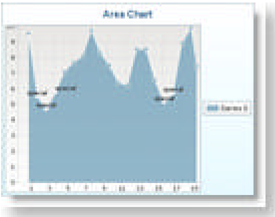
2d Grouping with Lines Chart

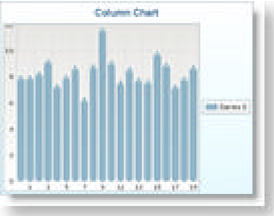
(Images\Grouping_Lines_Chart_Named_2D.png)

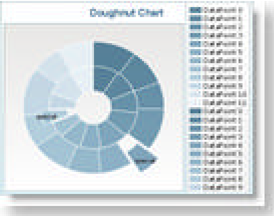


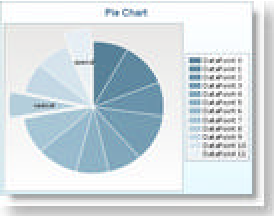


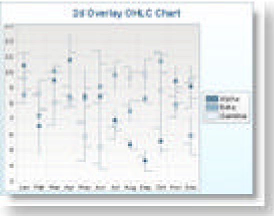


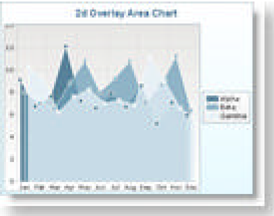


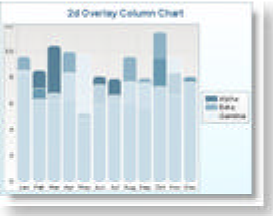


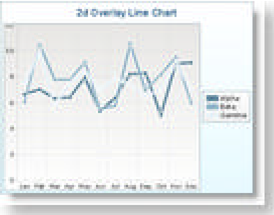




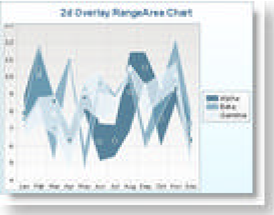


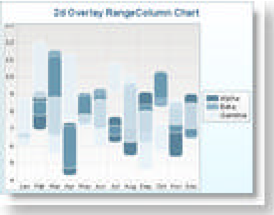


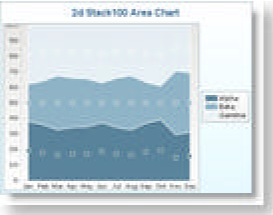


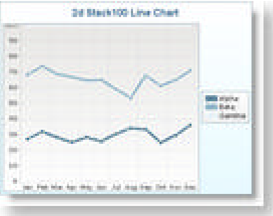


(Images\Overlay_Column_Chart_Named_2D.png)

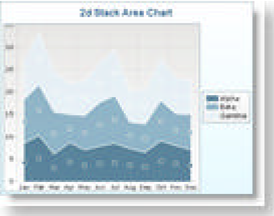


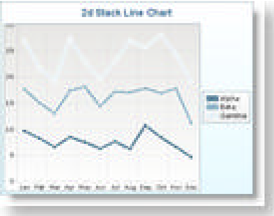


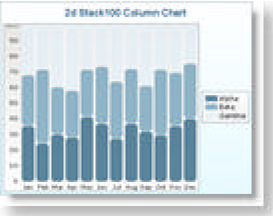




(Images\Stack_100_Area_Chart_Named_2D.png)
(Images\Stack_100_Line_Chart_Named_2D.png)

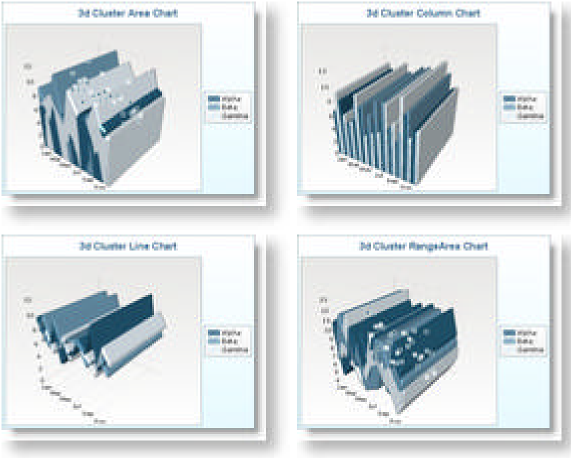




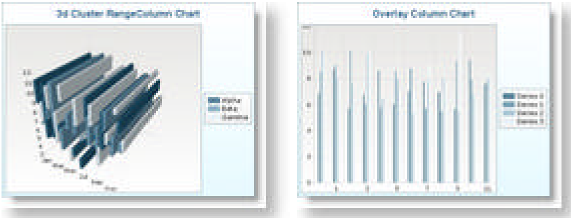


4.5.1.2.2 WebChart 3D Gallery

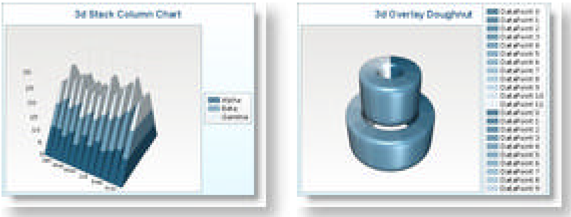
The screen shots below are some of the more popular 3D WebChart combinations.



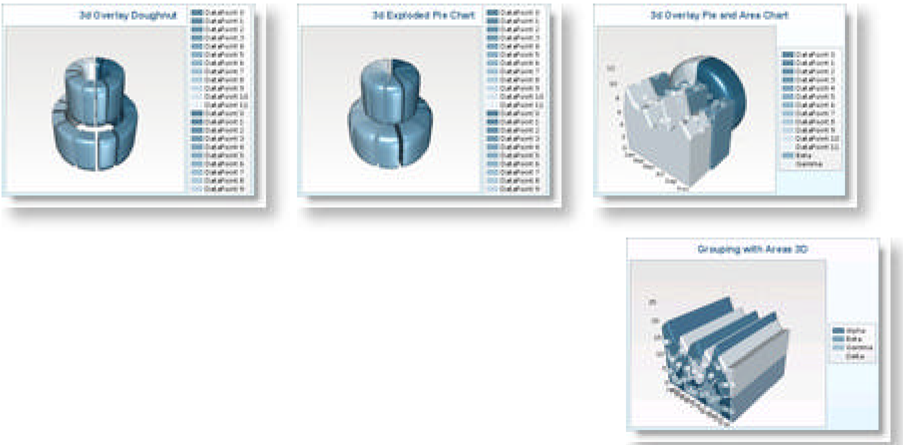
(Images\Cluster_Column_Chart_Named_3D.png)



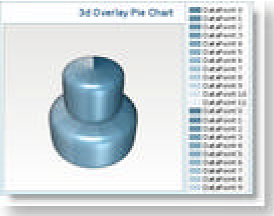
(Images\Column_Sizes_Named_3D.png)

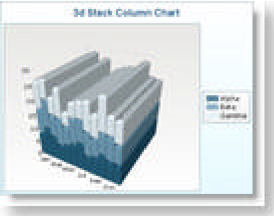


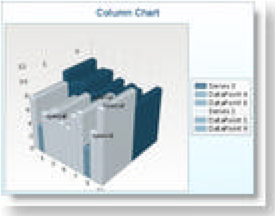
(Images\Doughnut_Chart_Named_3D.png)

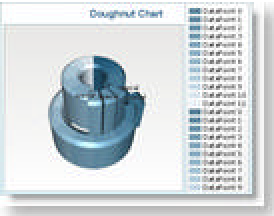


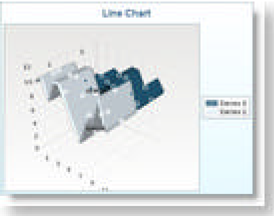
(Images\Overlay_Pie_Area_Chart_Named_3D.png)

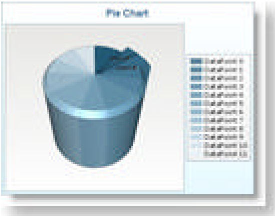


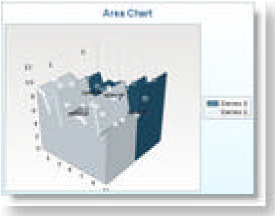


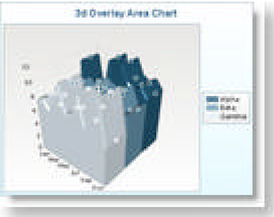


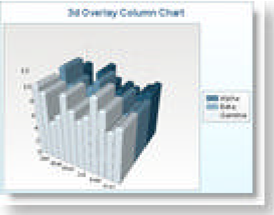


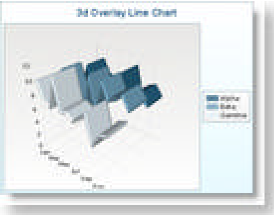




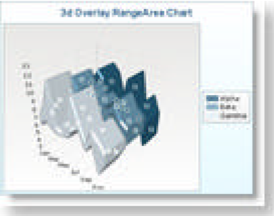


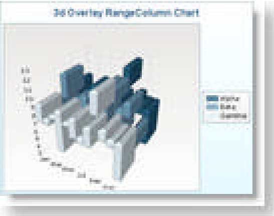




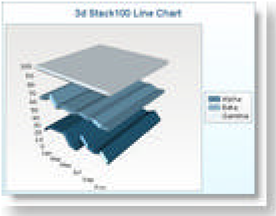


(Images\Overlay_Column_Chart_Named_3D.png)



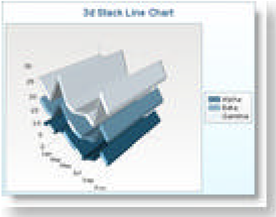


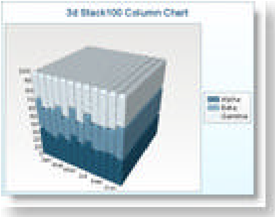
(Images\Stack_100_Area_Chart_Named_3D.png)



(Images\Stack_100_Line_Chart_Named_3D.png)

3d Stack Area Chart
Alpha
Beta
Gamma





4.5.1.3 Feature Overview

This topic offers you information about the various features available in the release of the NetAdvantage for JSF *WebChart*.

Over 20 Chart Types - There are over 20 chart types available ranging from the routine (column charts, pie charts) to the more sophisticated (candlestick charts, stack charts, cluster charts). For a detailed listing of the available charts, see the **Chart Types (Section 4.5.2)** section.

AJAX-enabled - Like other NetAdvantage for JSF components, WebChart is AJAX-enabled to allow you to perform partial refreshes of page content in response to user interaction with the chart.

Series Groups - Combine chart types to meet your requirements by grouping the data series together in ways such as: overlay, cluster, stack or stack with totals amounting to 100 percent. For more information on chart group types, see **Group Types. (Section 4.5.3)**

Data Binding - Data series can come from different data sources, and you can insert values for legends, markers and tooltips.

Smart Rendering - An intelligent rendering engine switches between the 2D Sun® graphics library, and the 3D Java™ bindings for OpenGL® (JOGL) API for hardware accelerated 3D graphics when supported.

Fully Customizable Look and Feel - You can easily style your chart using the CSS extensions available. For more information, see **Designing the Look and Feel (Section 5)**.

4.5.2 Chart Types

The *WebChart*™ component provides both two-dimensional and three-dimensional chart types. Clicking on the link below will provide you with information to help you understand how to create either a 2D or 3D chart.

- **2D Charts (Section 4.5.2.1)**
- **3D Charts (Section 4.5.2.2)**

Below is a full listing of our chart types. Depending on the type of chart that you want to use to display graphical data in your application, click the corresponding link so you can access key information about it.

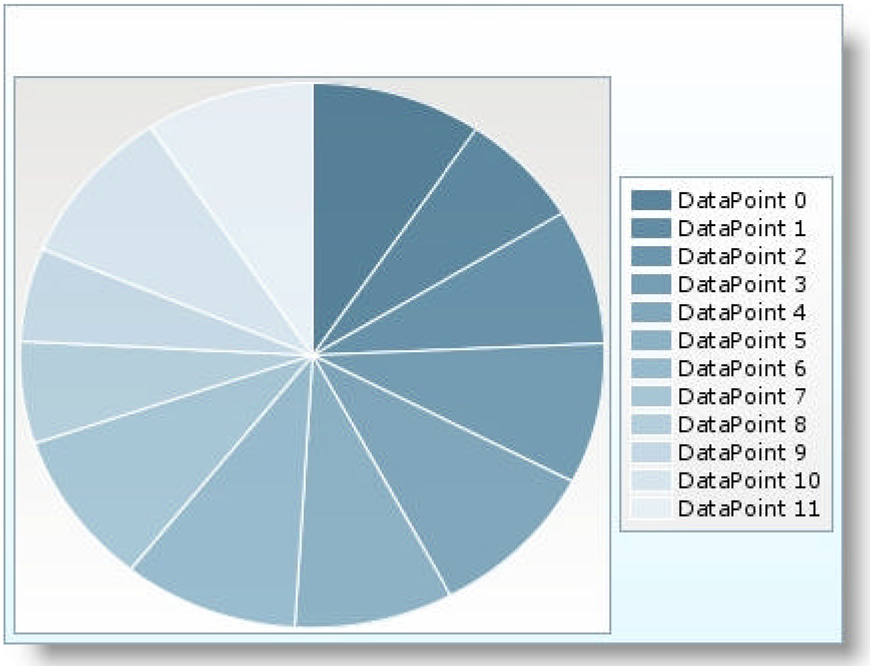
- **Area Chart (Section 4.5.2.3)**
- **Candlestick Chart (Section 4.5.2.4)**
- **Column Chart (Section 4.5.2.5)**
- **Doughnut Chart (Section 4.5.2.6)**
- **Line Chart (Section 4.5.2.7)**
- **OHLC Chart (Section 4.5.2.8)**
- **Pie Chart (Section 4.5.2.9)**
- **RangeArea Chart (Section 4.5.2.10)**
- **RangeColumn Chart (Section 4.5.2.11)**
- **Treemap Chart (Section 4.5.2.12)**

4.5.2.1 2D Charts

Two-dimensional and three-dimensional charts can be created using the *WebChart™* component. It is very easy to create both 2D and 3D WebCharts. Defining the "projectionType" attribute renders the WebChart either as a 2D WebChart or a 3D WebChart.

2D Charts

By setting the "projectionType" attribute to "2D" defines the chart as a 2D WebChart. No other attributes need to be changed.



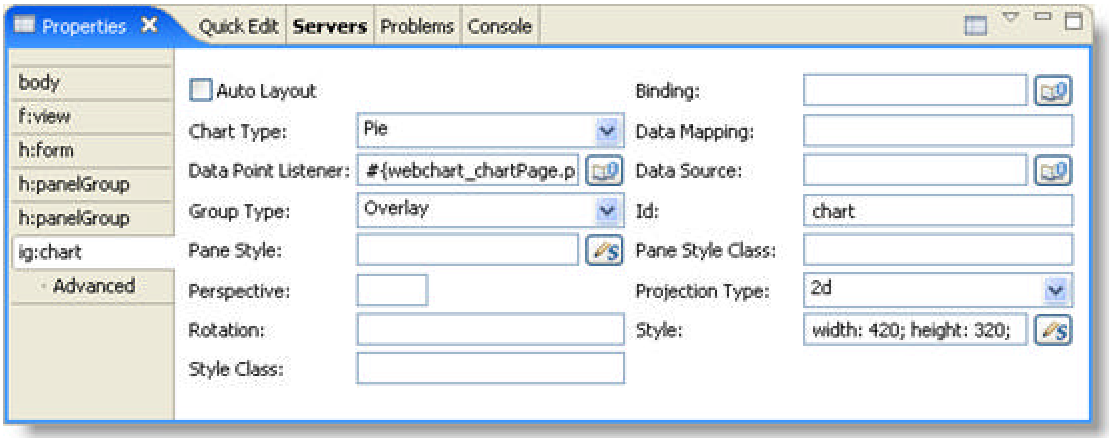
To create a 2D chart:

1. Set the <projectionType> attribute of the chart tag:
   In JSP:

```
<ig:chart id="chart"
    chartType="Pie" groupType="Overlay"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

To create a 2D chart in IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. In Design view, select the chart component on your page.
2. In the Properties tab, select ig:chart.
3. In the Projection Type drop down list, select 2d.



□ Auto Layout

Chart Type: Pie

Data Point Listener: #{webchart_chartPage.p

Group Type: Overlay

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4.5.2.2 3D Charts

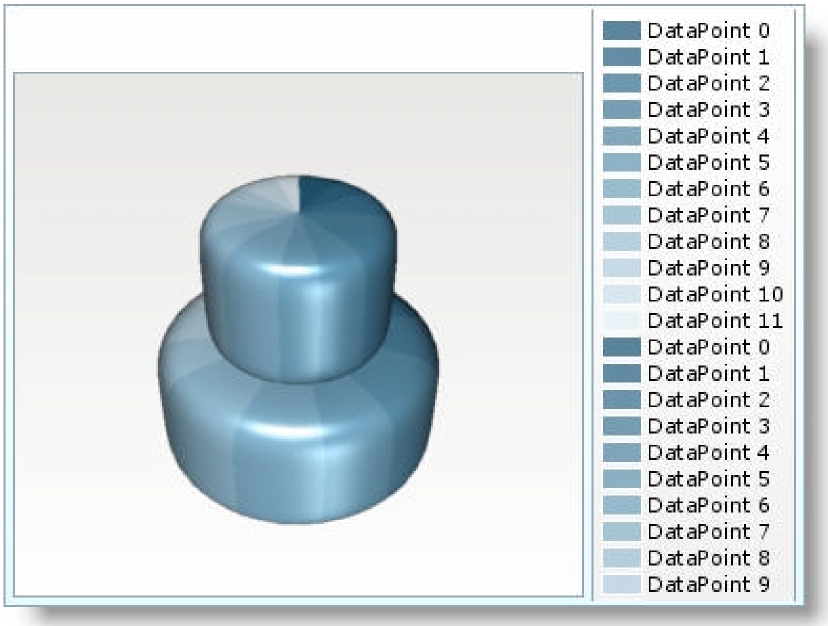
Two-dimensional and three-dimensional charts can be created using the *WebChart™* component. It is very easy to create both 2D and 3D WebCharts. Defining the "projectionType" attribute renders the WebChart either as a 2D WebChart or a 3D WebChart.

3D Charts

To make a 3D chart, you simply need to change the projectionType attribute to "3D" and add a rotation and perspective attribute.

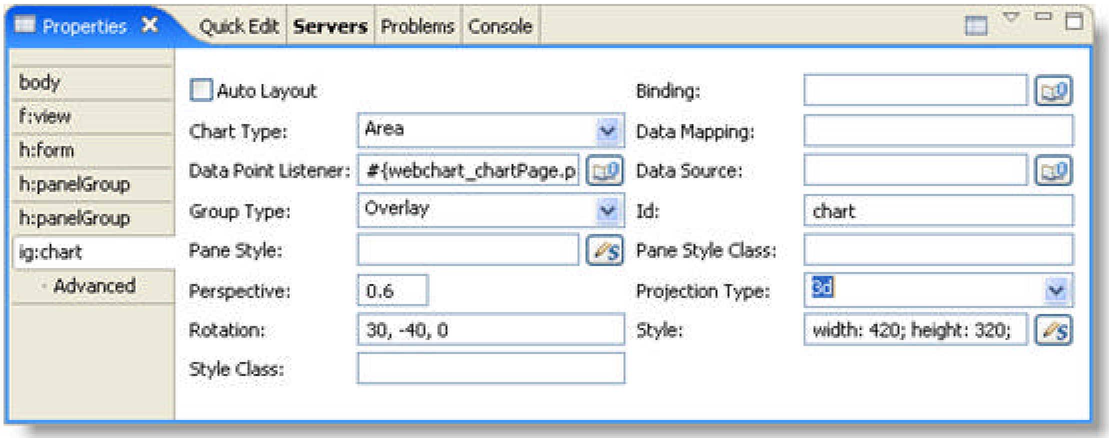
The rotation attribute is used to define the x, y and z rotation of the chart in the 3D space. Although the rotation is not required, if it is not specified the rotation will default to 0,0,0. This will effectively show the 3D chart in a 2D perspective.

The perspective attribute is used to provide a better representation of a 3D chart on a 2D surface. The lower the value, the more of a fisheye effect the chart will have. The range of the value is between 0 and 1.



To create a 3D chart in IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. In Design view, select the chart component on your page.
2. In the Properties tab, select ig:chart.
3. Set the following properties:
   - Projection Type -- 3d
   - Rotation -- 30, -40, 0
   - Perspective -- 0.6



□ Auto Layout

Chart Type:	Area	Binding:	
Data Point Listener:	#{webchart_chartPage.p	Data Mapping:	
Group Type:	Overlay	Data Source:	
Pane Style:		Id:	chart
Perspective:	0.6	Pane Style Class:	
Rotation:	30, -40, 0	Projection Type:	3d
Style Class:		Style:	width: 420; height: 320;

4.5.2.3 Area Chart

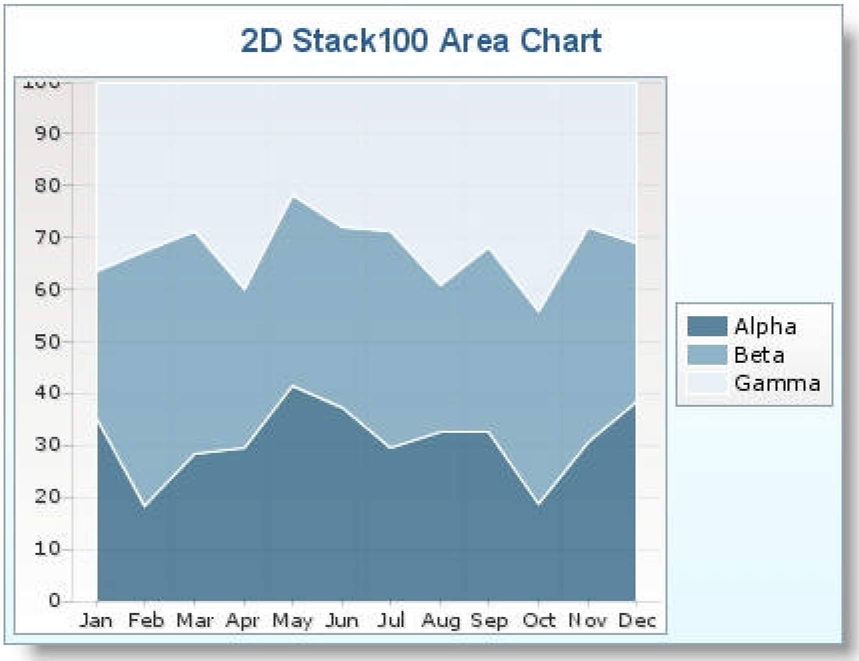
An area chart emphasizes the amount of change over a period of time or compares multiple items. An area chart also shows the relationship of parts to a whole by displaying the total of the plotted values.

An area chart displays series as a set of points connected by a line with the area below the line filled in. Values are represented on the y-axis and categories are displayed on the x-axis.

An area chart visually displays a change in values by filling in the portion of the graph beneath the line connecting various data points.

This topic will demonstrate how to create an area chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create an area chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- Area
   - groupType -- Stack100
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="Area"
    groupType="Stack100"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Stack 100 Area Chart", as shown in the JSP code below.
   In JSP:

```
<ig:caption
    position="top"
    caption="2d Stack100 Area Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:
    - type -- y
    - autoRange -- true
    - autoRangeSnap -- true
    - autoGridLines -- true

    In JSP:

    ```
    <ig:axis type="y"
        autoRange="true"
        autoRangeSnap="true"
        autoTickMarks="true"
        autoGridLines="true"/>
    ```

8. Set the series of your chart as shown in the JSP code below:
    In JSP:

    ```
    <ig:series
        legendCaption="#{webchart_numbersSource.name0}"
        dataMapping="value: column0; tooltipCaption: caption"
        dataSource="#{webchart_numbersSource.sparse0}" />

    <ig:series
        legendCaption="#{webchart_numbersSource.name1}"
        dataMapping="value: column0; tooltipCaption: caption"
        dataSource="#{webchart_numbersSource.sparse1}" />

    <ig:series
        legendCaption="#{webchart_numbersSource.name2}"
        dataMapping="value: column0; tooltipCaption: caption"
        dataSource="#{webchart_numbersSource.sparse2}" />
    ```

9. Close the <ig:chart> tag, as shown in the JSP code below.
    In JSP:

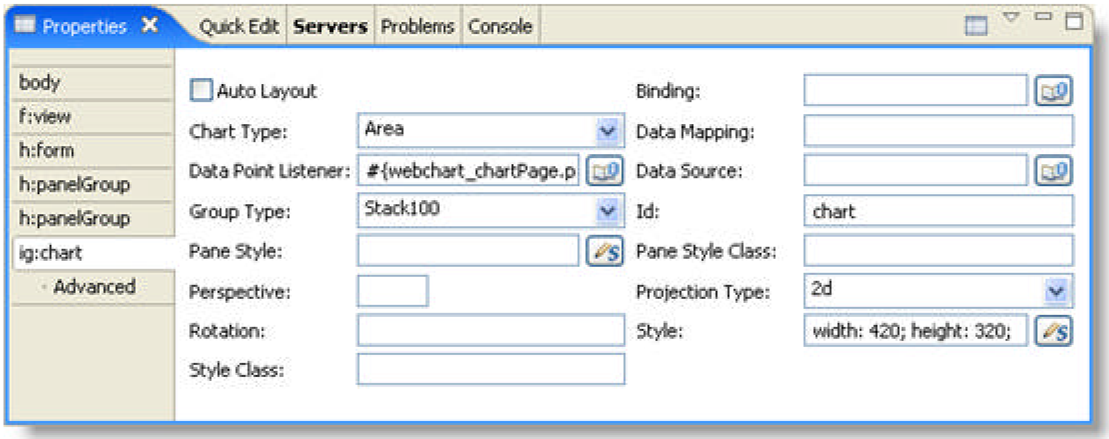
    ```
    </ig:chart>
    ```

10. Save and run the project.

To create an area chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
    - Chart Type -- Area
    - Data Point Listener -- #{webchart_chartPage.processDataPoint}
    - Group Type -- Stack100
    - Id -- chart

- Projection Type -- 2d
- Style -- width: 420; height: 320



☐ Auto Layout
Chart Type: Area
Data Point Listener: #{webchart_chartPage.p
Group Type: Stack100
Pane Style:
Perspective:
Rotation:
Style Class:
Binding:
Data Mapping:
Data Source:
Id: chart
Pane Style Class:
Projection Type: 2d
Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Stack100 Area Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x" autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true">
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y" autoRange="true"
    autoRangeSnap="true" autoTickMarks="true" autoGridLines="true"/>

<ig:series
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series
```

```
    legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

5. Save and run the project.

4.5.2.4 Candlestick Chart

Candlestick charts are often used to plot stock prices, and show the stock's high, low, open and close prices for each day.

Candlestick charts consists of visual elements called "Candles". Each candle has a "body" and a "wick".

The benefit of candlestick charts is that they signal indications of market turns earlier than other charts.

The length of the "body" of the candlestick chart indicates the span between a session's opening and closing value of an investment. If the closing value is lower than the opening value, the body is usually a dark color, such as black. But if the closing value is higher than the opening value, the body is usually a light color, such as white.

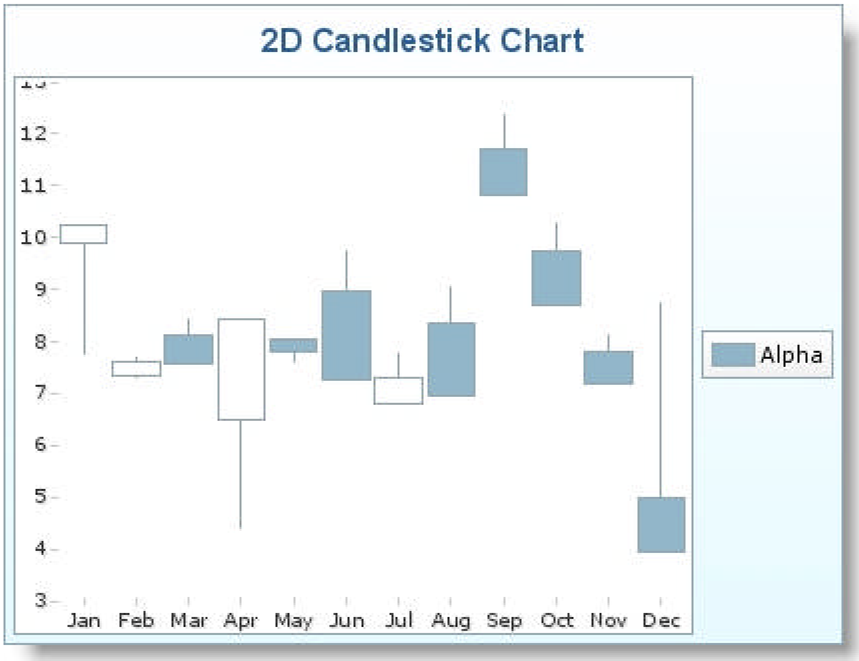
The lines above and below that body are called the "wicks". The top wick is the highest price during a session and the bottom wick is the lowest price during a session. Based on the color and the length of the body as well as the length of the wick, the trend in the market for particular investment can be determined.

For example, a long light-colored candle body indicates the investment's share prices are in an upswing signifying a buying trend, whereas a long dark-colored candle body indicates the investment's share prices are in a downswing.

Moving averages provide excellent clues as to whether a particular investment's share prices are moving in the upward or downward direction. It is calculated as the average of the last n days' prices. As the days progress on the time-scale, the oldest day is removed and the current day is added for new average calculation. Changes in trend can be inferred when two moving averages of different periods cross. Charts in these scenarios can have more than one moving average, and there is flexibility in whether the moving average is to be calculated on high, low, open or close values of the day.

This topic will demonstrate how to create a candlestick chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a candlestick chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:

- id -- chart
- chartType -- Candlestick
- projectionType -- 2d
- style -- width: 420; height: 320
- dataPointListener -- #{webchart_chartPage.processDataPoint}

In JSP:

```
<ig:chart
    id="chart"
    chartType="Candlestick"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Candlestick Chart", as shown in the JSP code below.
In JSP:

```
<ig:caption
    position="top"
    caption="2d Candlestick Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
```

```
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- y
   - autoRange -- true
   - autoRangeSnap -- true
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="y"
    autoRange="true"
    autoRangeSnap="true"
    autoTickMarks="true"
    autoGridLines="true"/>
```

8. Set the series of your chart as shown in the JSP code below:
   In JSP:

```
<ig:series legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="open: column1;
                 high: column2;
                 low: column3;
                 close: column4;
                 tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />
```

9. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

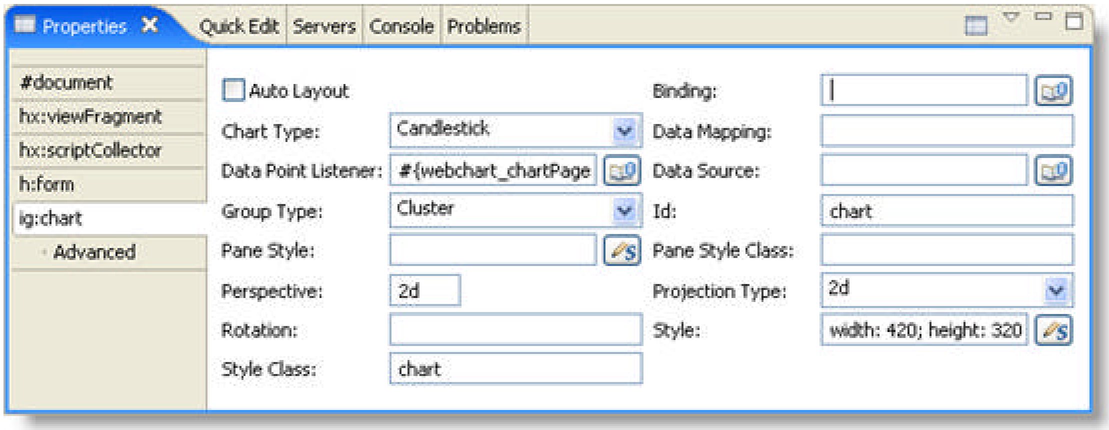
```
</ig:chart>
```

10. Save and run the project.

To create a candlestick chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.

3. Set the following properties:
    - Chart Type -- Candlestick
    - Data Point Listener -- #{webchart_chartPage.processDataPoint}
    - Id -- chart
    - Projection Type -- 2d
    - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Candlestick

Data Point Listener: #{webchart_chartPage

Group Type: Cluster

Pane Style:

Perspective: 2d

Rotation:

Style Class: chart

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Candlestick Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true" />

<ig:series legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="open: column1;
                 high: column2;
```

```
                low: column3;
                close: column4;
                tooltipCaption: caption"
        dataSource="#{webchart_numbersSource.sparse0}" />
```

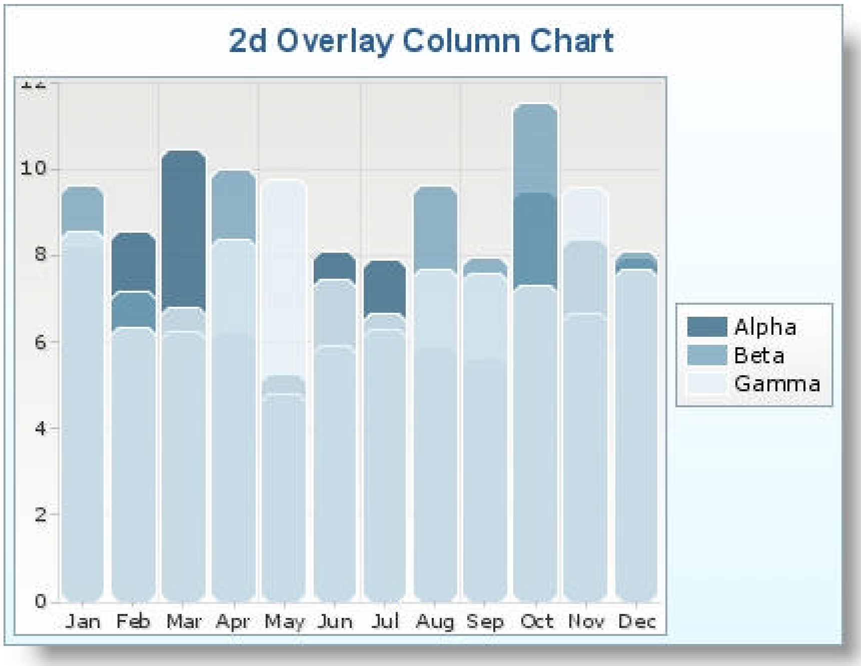
5. Save and run the project.

4.5.2.5 Column Chart

A column chart shows the changes in a data series over time or compares multiple items. Types of items are arranged horizontally and data values are plotted vertically to emphasize variation over time.

This topic will demonstrate how to create a column chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a column chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- Column
   - groupType -- Overlay
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="Column"
    groupType="Overlay"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Overlay Column Chart", as shown in the JSP code below.

In JSP:

```
<ig:caption
    position="top"
    caption="2d Overlay Column Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:

- type -- y
- autoRange -- true
- autoRangeSnap -- true
- autoGridLines -- true

In JSP:

```
<ig:axis type="y"
    autoRange="true"
    autoRangeSnap="true"
    autoTickMarks="true"
    autoGridLines="true"/>
```

8. Set the series of your chart as shown in the JSP code below:
   In JSP:

```
<ig:series legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

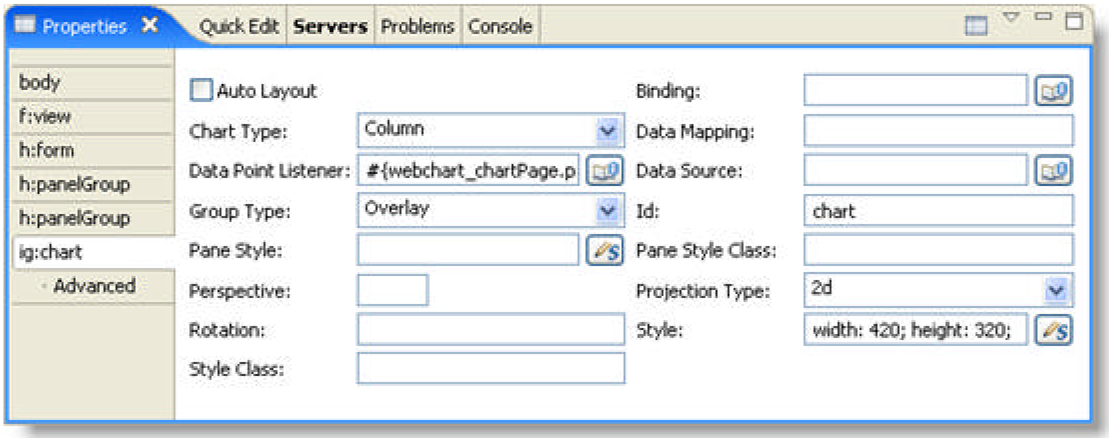
9. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

```
</ig:chart>
```

10. Save and run the project.

To create a column chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Column
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Group Type -- Overlay
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Column

Data Point Listener: #{webchart_chartPage.p

Group Type: Overlay

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
In JSP:

```
<ig:caption position="top" caption="2d Overlay Column Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true"/>

<ig:series legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

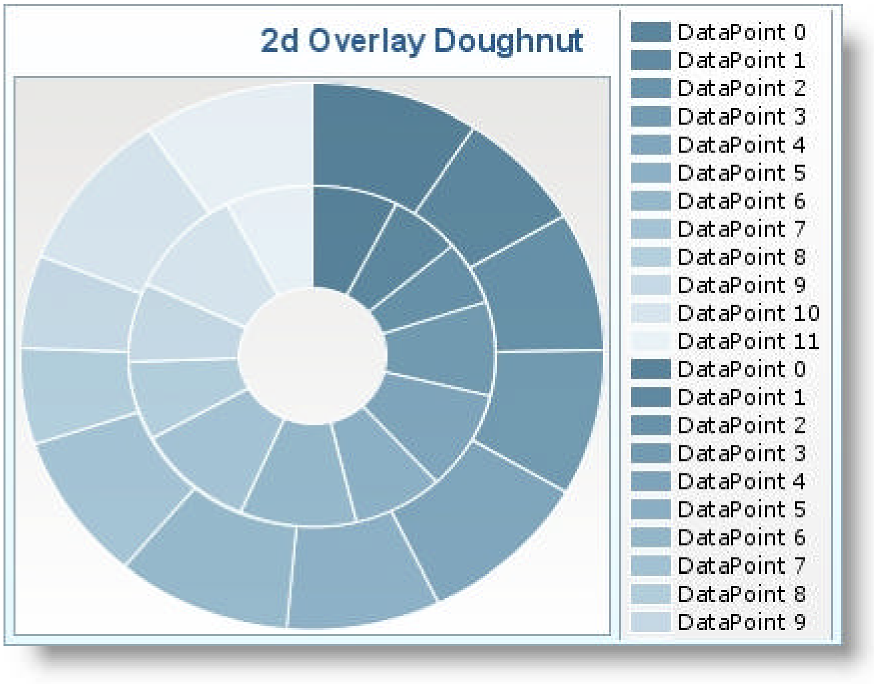
5. Save and run the project.

4.5.2.6 Doughnut Chart

A doughnut chart is a lot like a pie chart but it can hold multiple series. Like a pie chart, a doughnut chart shows the size of items that make up a data series proportional to the total of the items in the series. Doughnut charts display value data as percentages of the whole. Categories are represented by individual slices.

This topic will demonstrate how to create a doughnut chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a doughnut chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- Doughnut
   - groupType -- Overlay
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="Doughnut"
    groupType="Overlay"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Overlay Doughnut Chart", as shown in the JSP code below.
   In JSP:

   ```
   <ig:caption
       position="top"
       caption="2d Overlay Doughnut Chart"/>
   ```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

   ```
   <ig:legend position="right"
       autoItems="true" />
   ```

4. Set the series of your chart as shown in the JSP code below:
   In JSP:

   ```
   <ig:series legendCaption="#{webchart_numbersSource.name1}"
       dataMapping="value: column0; tooltipCaption: caption"
       dataSource="#{webchart_numbersSource.sparse1}"/>

   <ig:series legendCaption="#{webchart_numbersSource.name2}"
       dataMapping="value: column0; tooltipCaption: caption"
       dataSource="#{webchart_numbersSource.sparse2}" />
   ```

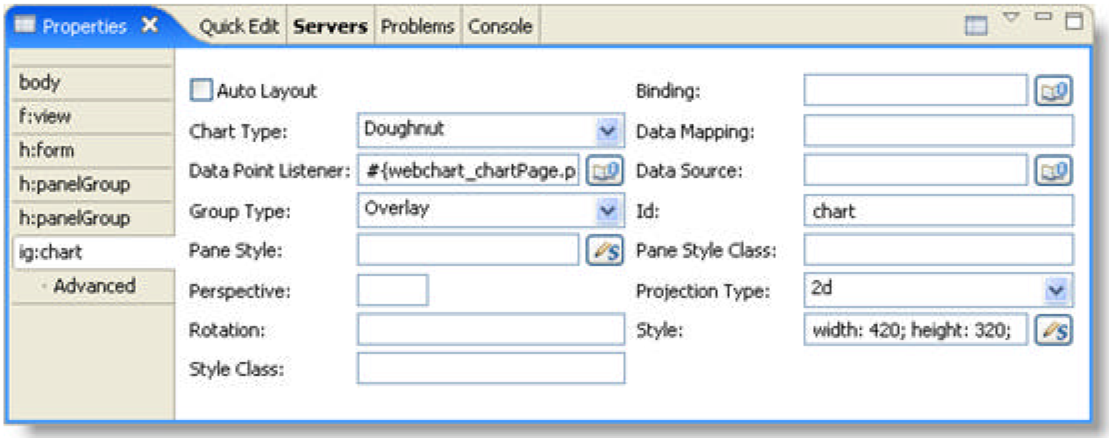
5. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

   ```
   </ig:chart>
   ```

6. Save and run the project.

To create a doughnut chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Doughnut
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Group Type -- Overlay
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



☐ Auto Layout

Chart Type:	Doughnut	Binding:	
Data Point Listener:	#{webchart_chartPage.p	Data Mapping:	
Group Type:	Overlay	Data Source:	
Pane Style:		Id:	chart
Perspective:		Pane Style Class:	
Rotation:		Projection Type:	2d
Style Class:		Style:	width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Overlay Doughnut Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:series legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}"/>

<ig:series legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

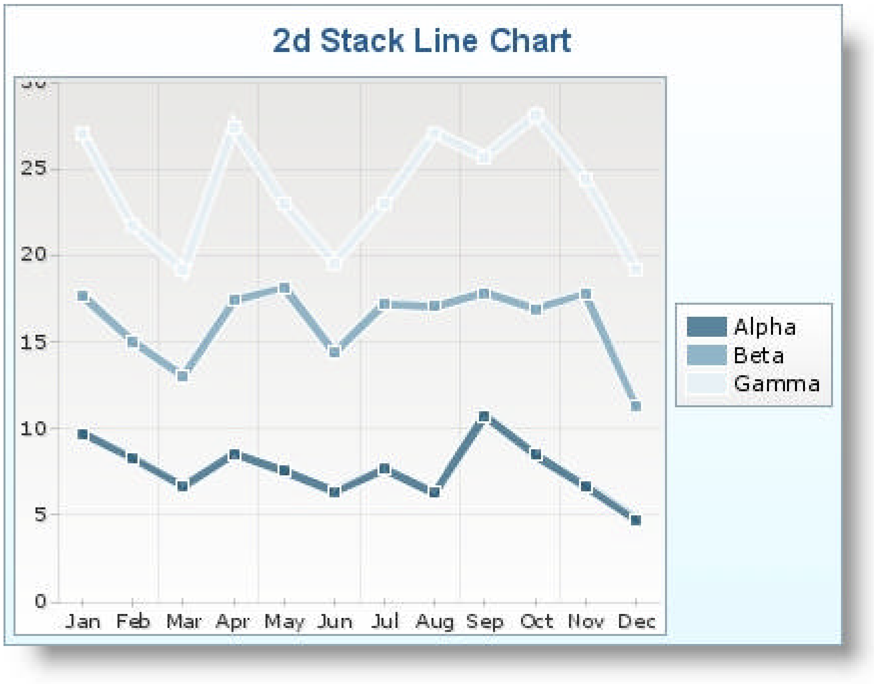
5. Save and run the project.

4.5.2.7 Line Chart

A line chart emphasizes the amount of change over a period of time or compares multiple items. Data points are plotted in series using evenly-spaced intervals and connected with a line to emphasize the relationships between the points.

This topic will demonstrate how to create a line chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a line chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
    - id -- chart
    - chartType -- Line
    - groupType -- Stack
    - projectionType -- 2d
    - style -- width: 420; height: 320
    - dataPointListener -- #{webchart_chartPage.processDataPoint}

    In JSP:

```
<ig:chart
    id="chart"
    chartType="Line"
    groupType="Stack"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Stack Line Chart", as shown in the JSP code below.
   In JSP:

```
<ig:caption
    position="top"
    caption="2d Stack Line Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- y
   - autoRange -- true
   - autoRangeSnap -- true
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="y"
    autoRange="true"
    autoRangeSnap="true"
    autoTickMarks="true"
    autoGridLines="true"/>
```

8. Set the series of your chart as shown in the JSP code below:
   In JSP:

```
<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

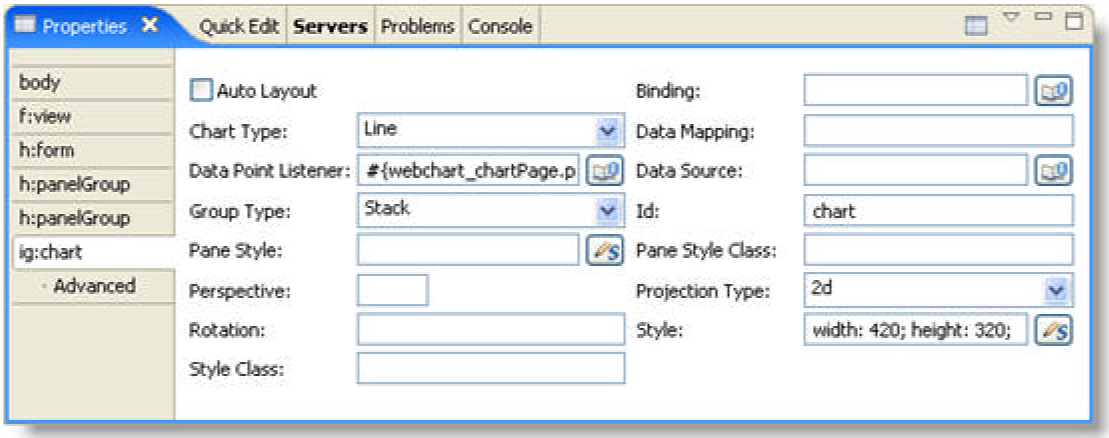
9. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

```
</ig:chart>
```

10. Save and run the project.

To create a line chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Line
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Group Type -- Stack
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Line

Data Point Listener: #{webchart_chartPage.p

Group Type: Stack

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:axis>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Stack Line Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true"/>

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
```

```
        dataMapping="value: column0; tooltipCaption: caption"
        dataSource="#{webchart_numbersSource.sparse2}" />
```

5. Save and run the project.

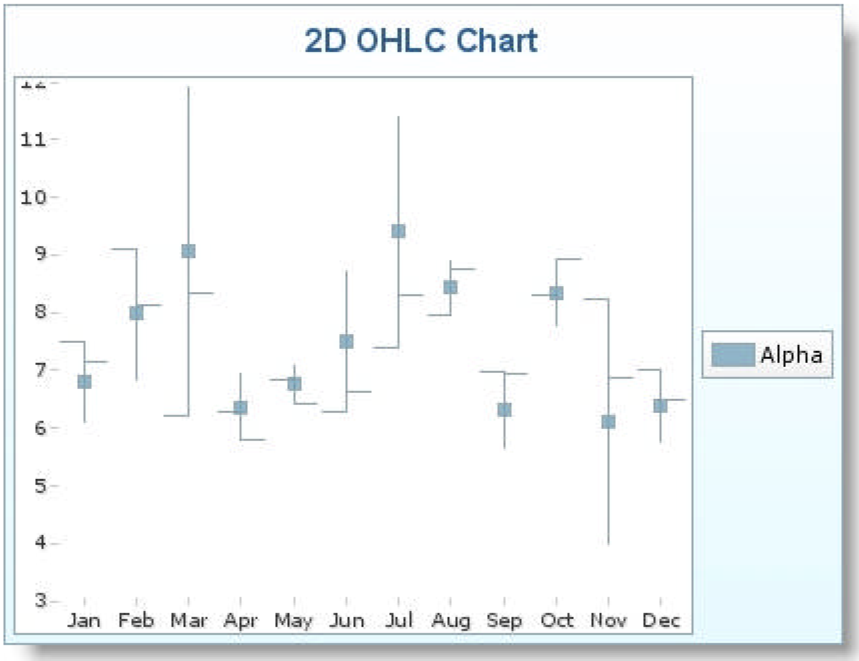
4.5.2.8 OHLC Chart

An Open, High, Low, Close Chart (OHLC Chart) is used for stocks to show the opening, high, low and closing prices for a stock. They are primarily used by technical analysts to spot trends and view stock movements particularly on a shorter-term basis.

OHLC charts are often used by analysts to spot trend and view stock movements, particularly on a shorter term basis.

This topic will demonstrate how to create a OHLC chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a OHLC chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- OHLC
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="OHLC"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d OHLC Chart", as shown in the JSP code below.
   In JSP:

```
<ig:caption
    position="top"
    caption="2d OHLC Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- y
   - autoRange -- true
   - autoRangeSnap -- true
   - autoGridLines -- true

   In JSP:

   ```
   <ig:axis type="y"
       autoRange="true"
       autoRangeSnap="true"
       autoTickMarks="true"
       autoGridLines="true"/>
   ```

8. Set the series of your chart as shown in the JSP code below:
   In JSP:

   ```
   <ig:series legendCaption="#{webchart_numbersSource.name0}"
       dataMapping="open: column1;
                    high: column2;
                    low: column3;
                    close: column4;
                    tooltipCaption: caption"
       dataSource="#{webchart_numbersSource.sparse0}" />
   ```

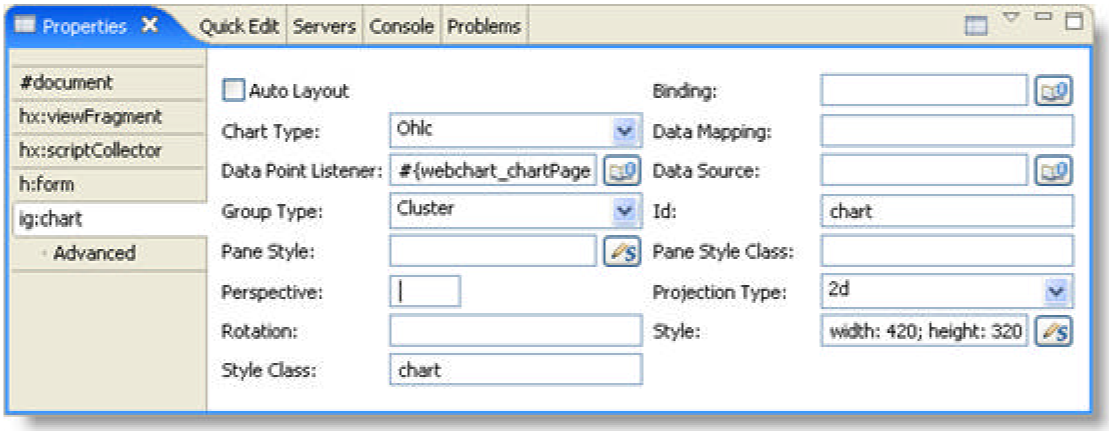
9. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

   ```
   </ig:chart>
   ```

10. Save and run the project.

To create a OHLC chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- OHLC
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



☐ Auto Layout

Chart Type: Ohlc

Data Point Listener: #{webchart_chartPage

Group Type: Cluster

Pane Style:

Perspective:

Rotation:

Style Class: chart

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:axis>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d OHLC Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true" />

<ig:series legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="open: column1;
                 high: column2;
                 low: column3;
                 close: column4;
                 tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}" />
```

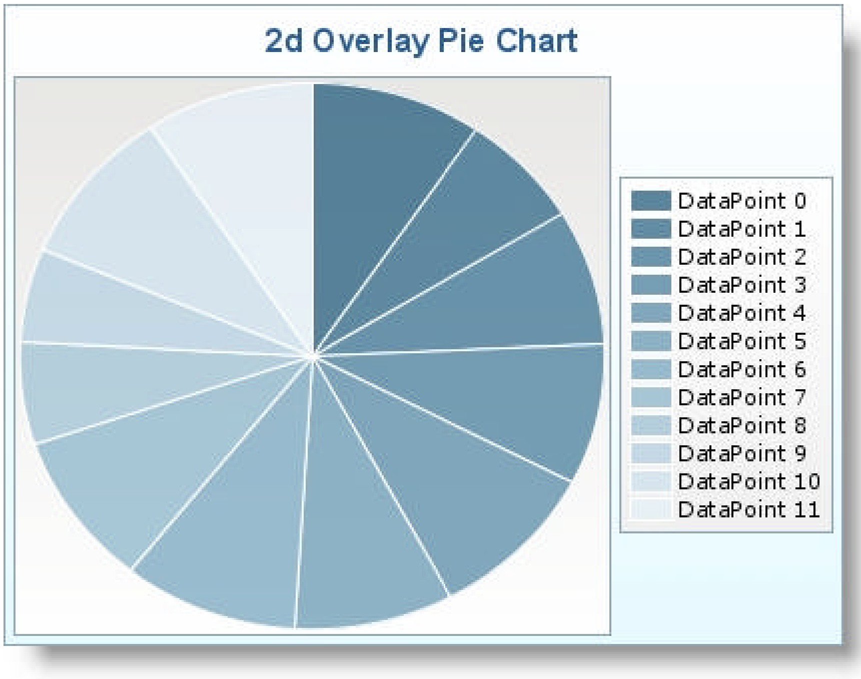
5. Save and run the project.

4.5.2.9 Pie Chart

A pie chart is a circular chart divided into sectors proportional to the percentages of the whole. A pie chart always shows a single data series, and is useful for determining which items or items in the series are most significant.

This topic will demonstrate how to create a pie chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a pie chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- Pie
   - groupType -- Overlay
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="Pie"
    groupType="Overlay"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Overlay Pie Chart", as shown in the JSP code below.
   In JSP:

```
<ig:caption
    position="top"
    caption="2d Overlay Pie Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the series of your chart as shown in the JSP code below:
   In JSP:

```
<ig:series legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}"/>
```

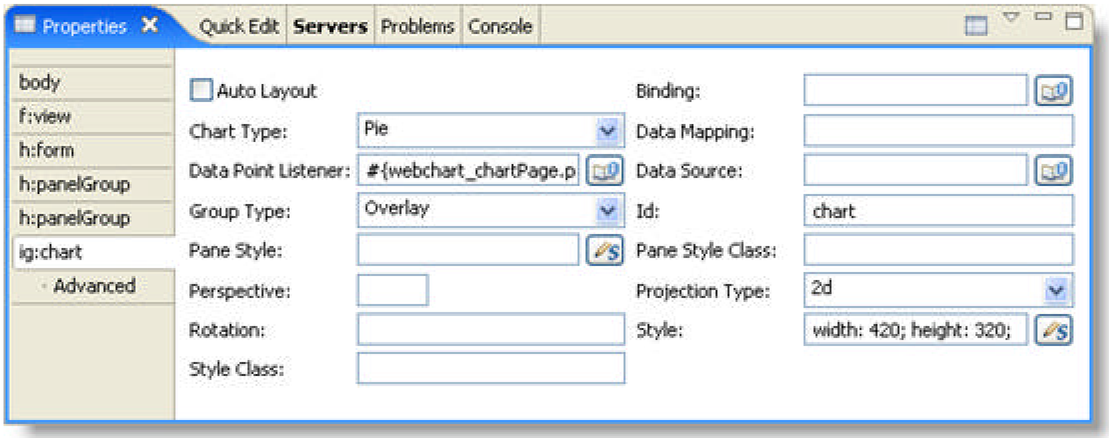
5. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

```
</ig:chart>
```

6. Save and run the project.

To create a pie chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Pie
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Group Type -- Overlay
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



☐ Auto Layout

Chart Type: Pie

Data Point Listener: #{webchart_chartPage.p

Group Type: Overlay

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Overlay Pie Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:series legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="value: column0; tooltipCaption: caption"
    dataSource="#{webchart_numbersSource.sparse0}"/>
```

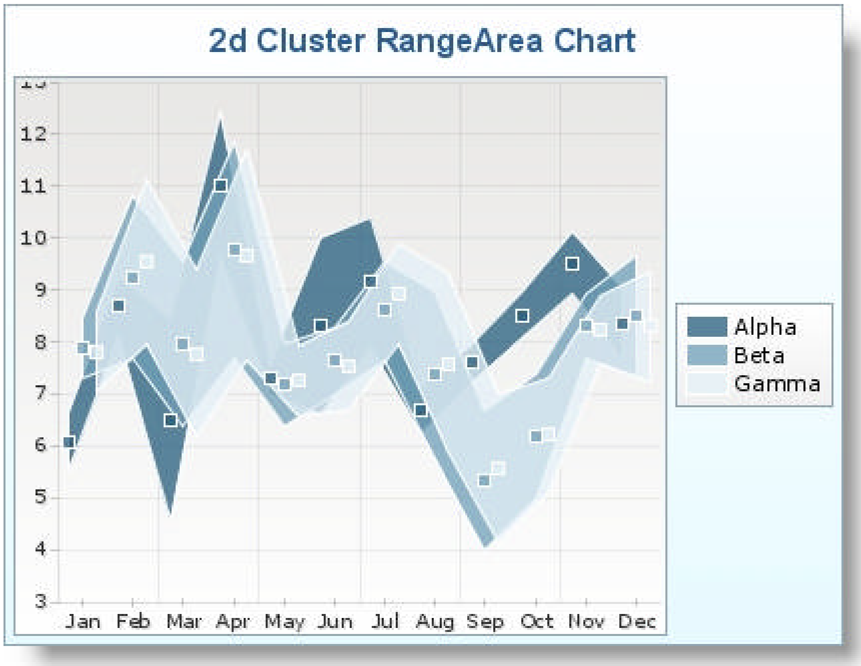
5. Save and run the project.

4.5.2.10 Range Area Chart

A range area chart behaves very much like a regular Area Chart, with the exception that both the bottom (low) and top (high) value of a range are plotted.

This topic will demonstrate how to create a range area chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a range area chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - chartType -- RangeArea
   - groupType -- Cluster
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    chartType="RangeArea"
    groupType="Cluster"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Cluster RangeArea Chart", as shown in the JSP code below.

In JSP:

```
<ig:caption
    position="top"
    caption="2d Cluster RangeArea Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:

- type -- y
- autoRange -- true
- autoRangeSnap -- true
- autoGridLines -- true

In JSP:

```
<ig:axis type="y"
    autoRange="true"
    autoRangeSnap="true"
    autoTickMarks="true"
    autoGridLines="true"/>
```

8. Set the series of your chart as shown in the JSP code below:
In JSP:

```
<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

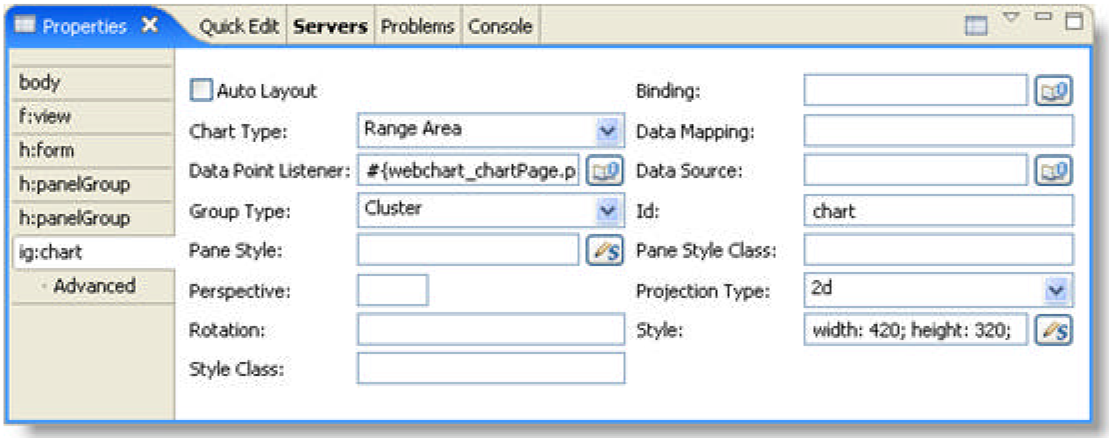
9. Close the <ig:chart> tag, as shown in the JSP code below.
In JSP:

```
</ig:chart>
```

10. Save and run the project.

To create a range area chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
    - Chart Type -- RangeArea
    - Data Point Listener -- #{webchart_chartPage.processDataPoint}
    - Group Type -- Cluster
    - Id -- chart
    - Projection Type -- 2d
    - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Range Area

Data Point Listener: #{webchart_chartPage.p

Group Type: Cluster

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:axis>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Cluster RangeArea Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true"/>

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
```

```
        dataMapping="low: column3; high: column2"
        dataSource="#{webchart_numbersSource.sparse2}" />
```

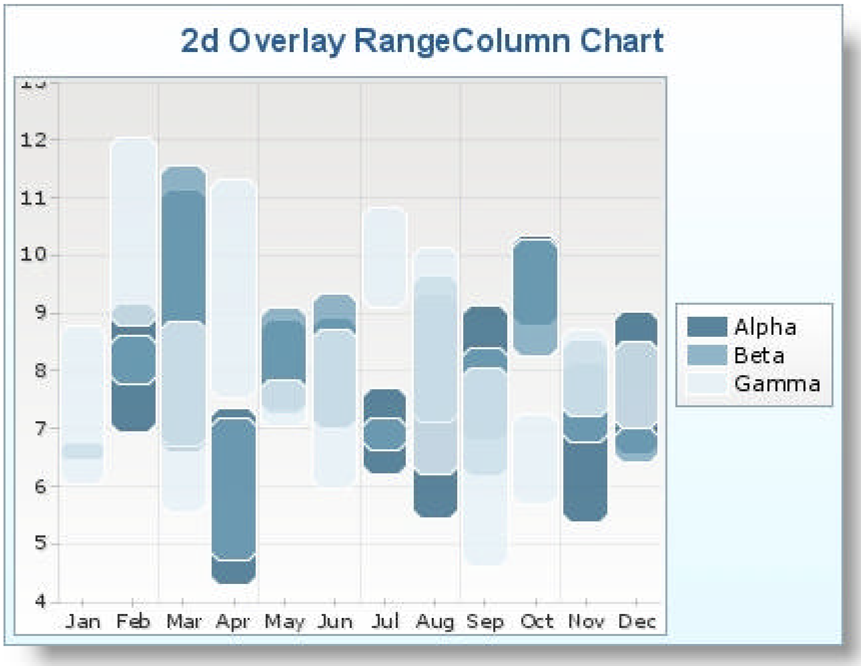
5. Save and run the project.

4.5.2.11 Range Column Chart

A range column chart behaves very much like a regular column chart, with the exception that both the bottom (low) and top (high) value of a range are plotted by "floating" the data point's column.

This topic will demonstrate how to create a range column chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a range column chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
    - id -- chart
    - chartType -- RangeColumn
    - groupType -- Overlay
    - projectionType -- 2d
    - style -- width: 420; height: 320
    - dataPointListener -- #{webchart_chartPage.processDataPoint}

    In JSP:

```
<ig:chart
    id="chart"
    chartType="RangeColumn"
    groupType="Overlay"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "2d Overlay RangeColumn Chart", as shown in the JSP code below.

In JSP:

```
<ig:caption
    position="top"
    caption="2d Overlay RangeColumn Chart"/>
```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:
   - position -- right
   - autoItems -- true

   In JSP:

```
<ig:legend position="right"
    autoItems="true" />
```

4. Set the x-axis of your chart with the following attributes, as shown in the JSP code below:
   - type -- x
   - autoRange -- true
   - autoRangeSnap -- false
   - autoTickMarks -- false
   - autoGridLines -- true

   In JSP:

```
<ig:axis type="x"
    autoRange="true"
    autoRangeSnap="false"
    autoTickMarks="false"
    autoGridLines="true">
```

5. Set the tick marks of your chart to display the months of the year, as shown in the JSP code below:
   In JSP:

```
<ig:tickMark value="0" tickCaption="Jan" />
<ig:tickMark value="1" tickCaption="Feb" />
<ig:tickMark value="2" tickCaption="Mar" />
<ig:tickMark value="3" tickCaption="Apr" />
<ig:tickMark value="4" tickCaption="May" />
<ig:tickMark value="5" tickCaption="Jun" />
<ig:tickMark value="6" tickCaption="Jul" />
<ig:tickMark value="7" tickCaption="Aug" />
<ig:tickMark value="8" tickCaption="Sep" />
<ig:tickMark value="9" tickCaption="Oct" />
<ig:tickMark value="10" tickCaption="Nov" />
<ig:tickMark value="11" tickCaption="Dec" />
```

6. Close the <ig:axis> tag, as shown in the JSP code below.
   In JSP:

```
</ig:axis>
```

7. Set the y-axis of your chart with the following attributes, as shown in the JSP code below:

- type -- y
- autoRange -- true
- autoRangeSnap -- true
- autoGridLines -- true

In JSP:

```
<ig:axis type="y"
    autoRange="true"
    autoRangeSnap="true"
    autoTickMarks="true"
    autoGridLines="true"/>
```

8. Set the series of your chart as shown in the JSP code below:
In JSP:

```
<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse2}" />
```

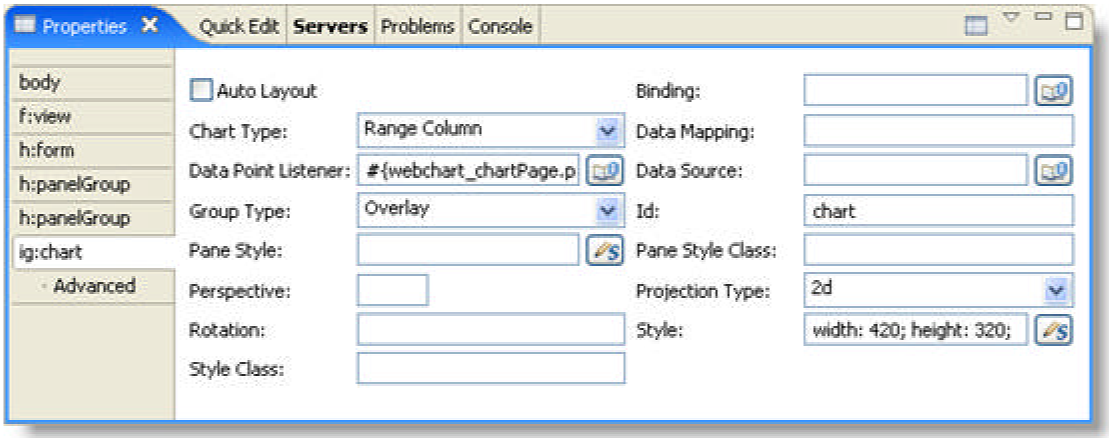
9. Close the <ig:chart> tag, as shown in the JSP code below.
In JSP:

```
</ig:chart>
```

10. Save and run the project.

To create a range column chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Range Column
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Group Type -- Overlay
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Range Column

Data Point Listener: #{webchart_chartPage.p

Group Type: Overlay

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:axis>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="2d Overlay RangeColumn Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:axis type="x"
    autoRange="true" autoRangeSnap="false"
    autoTickMarks="false" autoGridLines="true" >
    <ig:tickMark value="0" tickCaption="Jan" />
    <ig:tickMark value="1" tickCaption="Feb" />
    <ig:tickMark value="2" tickCaption="Mar" />
    <ig:tickMark value="3" tickCaption="Apr" />
    <ig:tickMark value="4" tickCaption="May" />
    <ig:tickMark value="5" tickCaption="Jun" />
    <ig:tickMark value="6" tickCaption="Jul" />
    <ig:tickMark value="7" tickCaption="Aug" />
    <ig:tickMark value="8" tickCaption="Sep" />
    <ig:tickMark value="9" tickCaption="Oct" />
    <ig:tickMark value="10" tickCaption="Nov" />
    <ig:tickMark value="11" tickCaption="Dec" />
</ig:axis>

<ig:axis type="y"
    autoRange="true" autoRangeSnap="true"
    autoTickMarks="true" autoGridLines="true"/>

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name0}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse0}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name1}"
    dataMapping="low: column3; high: column2"
    dataSource="#{webchart_numbersSource.sparse1}" />

<ig:series markerBulletVisible="true"
    legendCaption="#{webchart_numbersSource.name2}"
```

```
        dataMapping="low: column3; high: column2"
        dataSource="#{webchart_numbersSource.sparse2}" />
```

5. Save and run the project.

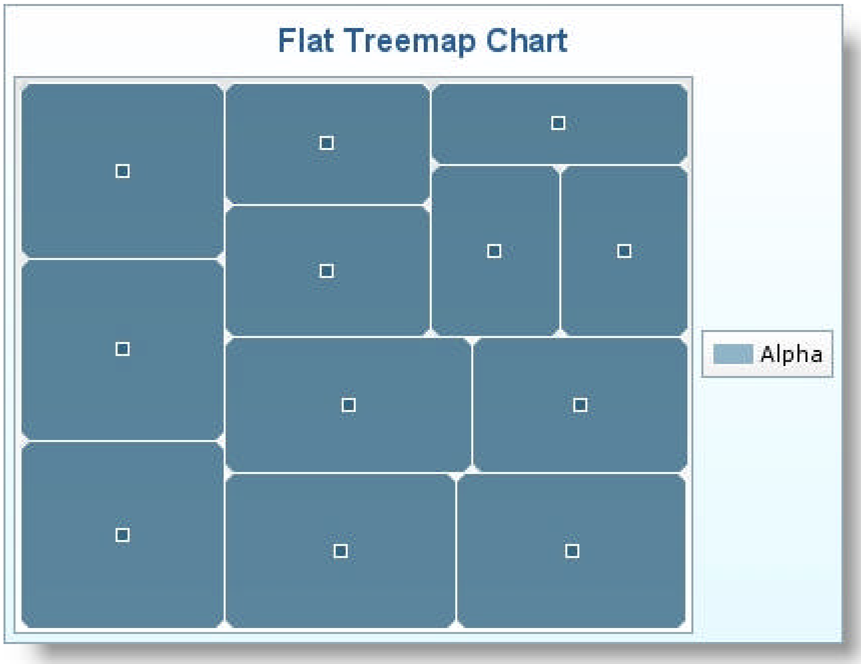
4.5.2.12 Treemap Chart

Treemap charts are used to display large hierarchical data sets in a rectangular space, which is divided into regions and then each region is divided again for each level of the hierarchy. You can identify the data on a treemap chart by the size and color of the region.

Treemap charts are typically used to visually the contents of a hard drive with thousands of files in 5 - 15 levels of directories.

This topic will demonstrate how to create a treemap chart, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a treemap chart using code:

1. Open the <ig:chart> tag and set the following attributes, as shown in the JSP code below:
   - id -- chart
   - binding -- #{webchart_chartPage.chartComponent}
   - chartType -- treeMap
   - projectionType -- 2d
   - style -- width: 420; height: 320
   - dataPointListener -- #{webchart_chartPage.processDataPoint}

   In JSP:

```
<ig:chart
    id="chart"
    binding="#{webchart_chartPage.chartComponent}"
    chartType="treeMap"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

2. Set the caption of your chart to "Flat Treemap Chart", as shown in the JSP code below.
   In JSP:

   ```
   <ig:caption
       position="top"
       caption="Flat Treemap Chart"/>
   ```

3. Set the legend of your chart with the following attributes as shown in the JSP code below:

   - position -- right
   - autoItems -- true

   In JSP:

   ```
   <ig:legend position="right"
       autoItems="true" />
   ```

4. Set the series of your chart as shown in the JSP code below:
   In JSP:

   ```
   <ig:series legendCaption ="#{webchart_numbersSource.name0}"
       markerBulletVisible="true"
       markerCaptionVisible="true"
       dataSource="value: column0;
                   tooltipCaption:caption"
                   dataSource="#{webchart_numbersSource.sparse0}" />
   ```

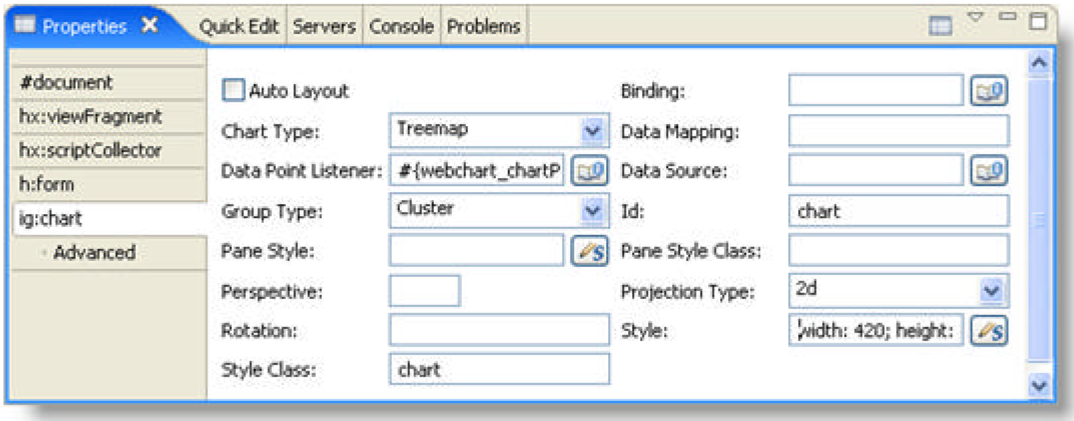
5. Close the <ig:chart> tag, as shown in the JSP code below.
   In JSP:

   ```
   </ig:chart>
   ```

6. Save and run the project.

To create a treemap chart using RAD:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Treemap
   - Data Point Listener -- #{webchart_chartPage.processDataPoint}
   - Id -- chart
   - Projection Type -- 2d
   - Style -- width: 420; height: 320



□ Auto Layout

Chart Type: Treemap

Data Point Listener: #{webchart_chartP

Group Type: Cluster

Pane Style:

Perspective:

Rotation:

Style Class: chart

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height:

4. Create other tags in JSP such as <ig:caption>, <ig:legend>, <ig:axis>, <ig:tickMark> and <ig:series> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption position="top" caption="Flat Treemap Chart"/>

<ig:legend position="right" autoItems="true" />

<ig:series
    legendCaption ="#{webchart_numbersSource.name0}"
    markerBulletVisible="true"
    markerCaptionVisible="true"
    dataSource="value: column0; tooltipCaption:caption"
    dataSource="#{webchart_numbersSource.sparse0}" />
```

5. Save and run the project.

4.5.3 Group Types

The data on a *WebChart*™ can also be grouped into different types of arrangements to better visualize the relationship between the different data sets.

The following is a list of the different group types that can be used within the WebChart component.

- **Cluster Groups (Section 4.5.3.1)**
- **Stack Groups (Section 4.5.3.2)**
- **Stack 100 Groups (Section 4.5.3.3)**
- **Overlay Groups (Section 4.5.3.4)**

Clicking on the link below will provide you with a short task based topic that explains how to set the group type of a chart.

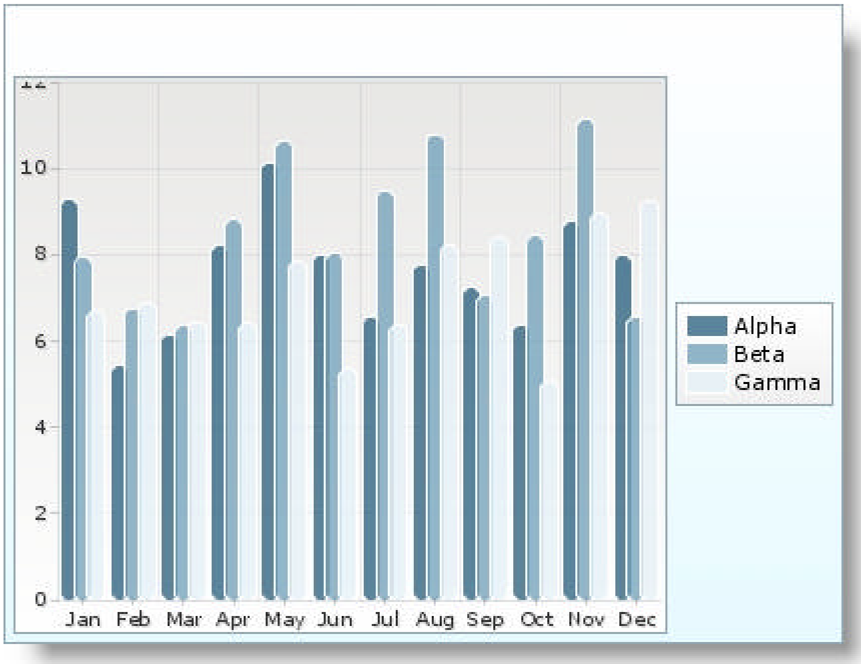
- **Set Group Type of the Chart (Section 4.5.3.5)**

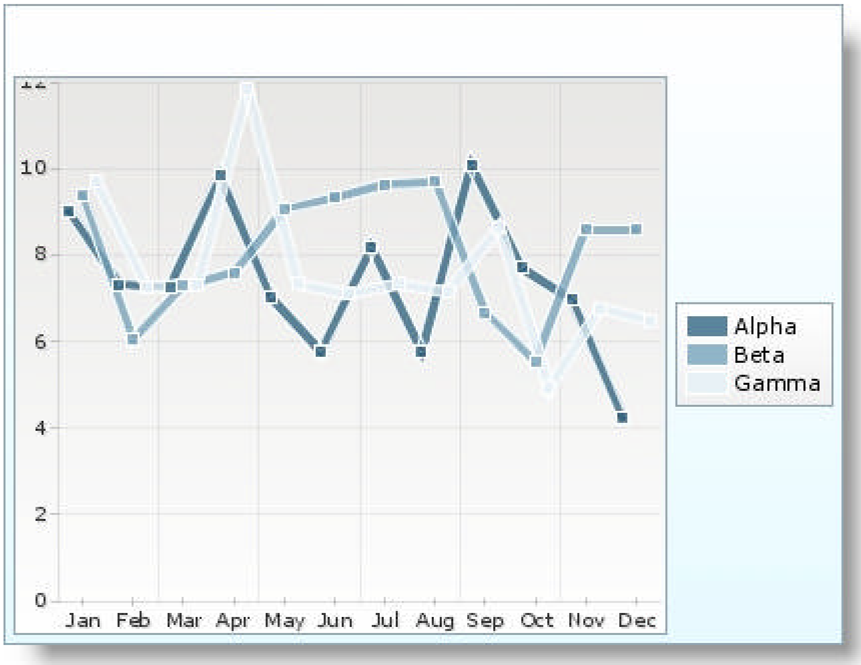
4.5.3.1 Cluster Groups

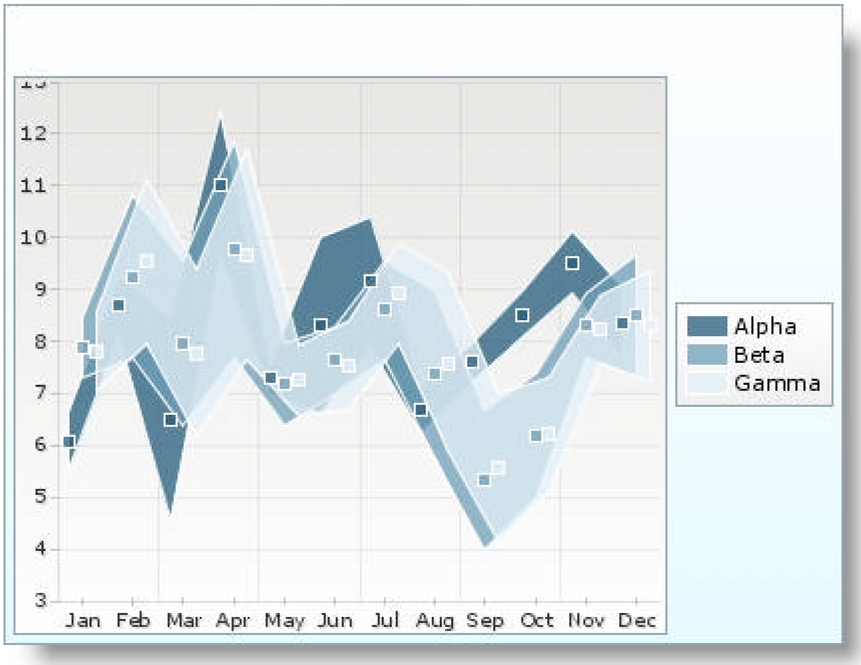
Cluster groups displays charts with the values of each individual item clustered together within one chart.

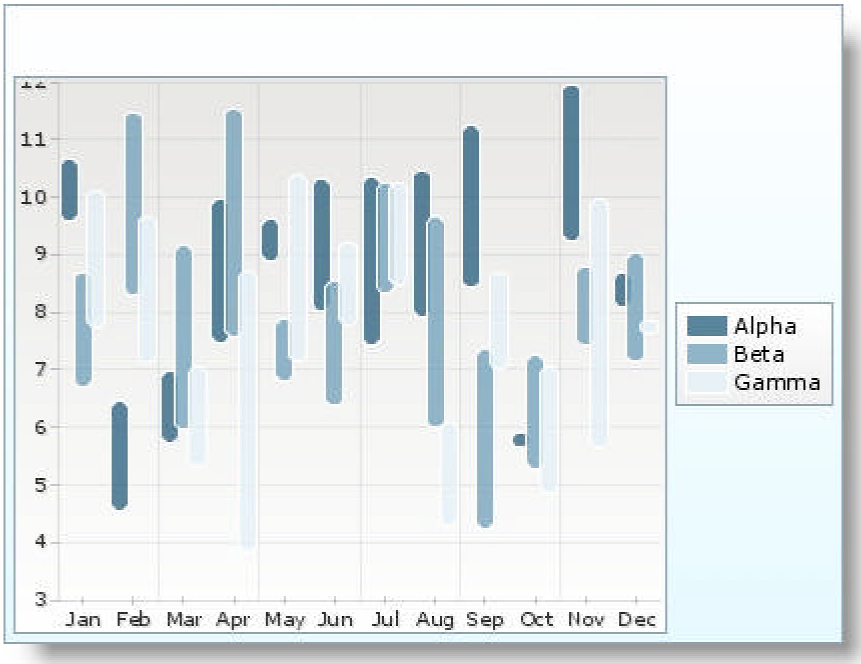
To create a clustered chart, set the groupType attribute to "cluster".

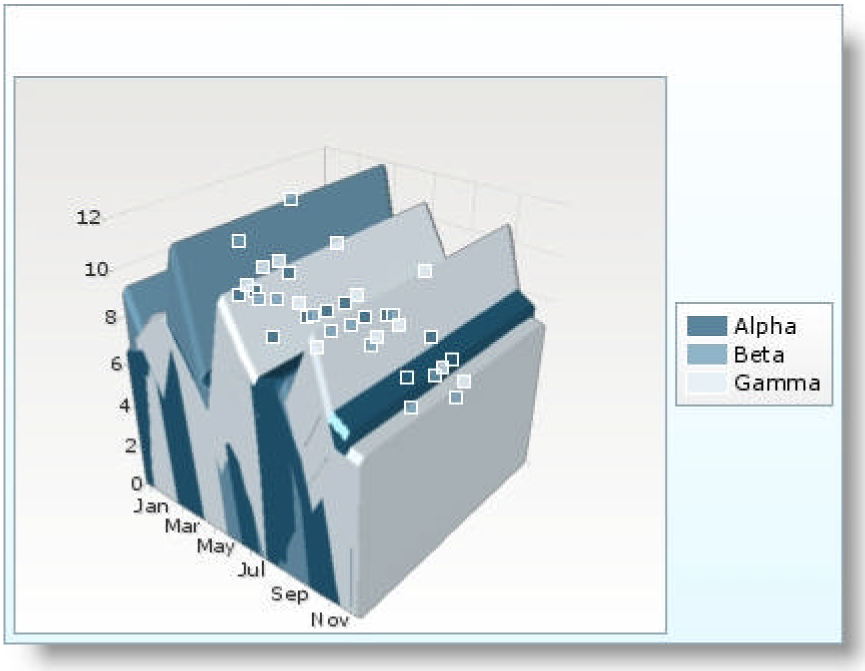
Below are examples of clustered charts.

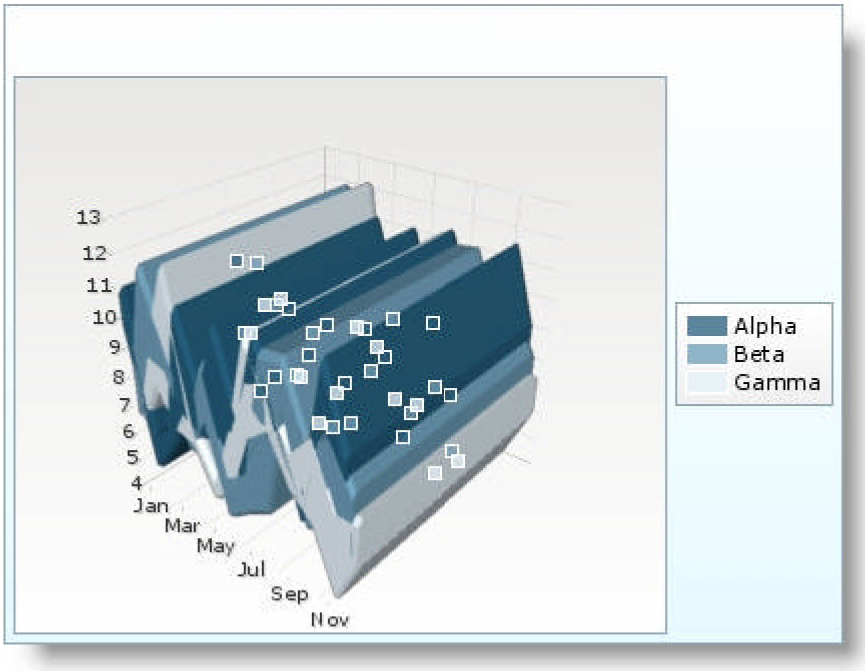


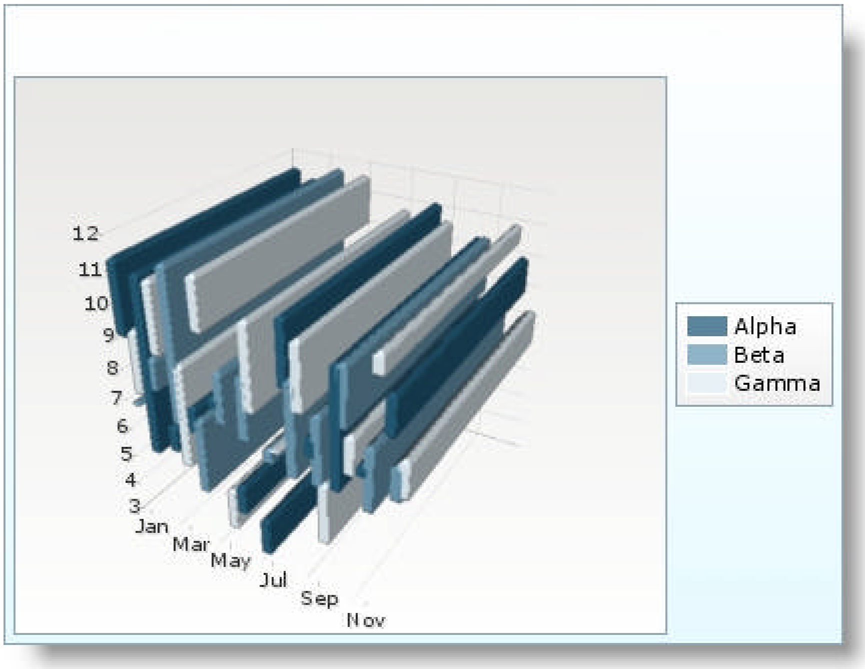



12
11
10
9
8
7
6
5
4
3
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Alpha
Beta
Gamma


11
10
9
8
7
6
5
4
3
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Alpha
Beta
Gamma


12
10
8
6
4
2
0
Jan
Mar
May
Jul
Sep
Nov
Alpha
Beta
Gamma


13
12
11
10
9
8
7
6
5
4
Jan
Mar
May
Jul
Sep
Nov
Alpha
Beta
Gamma



Related Topic

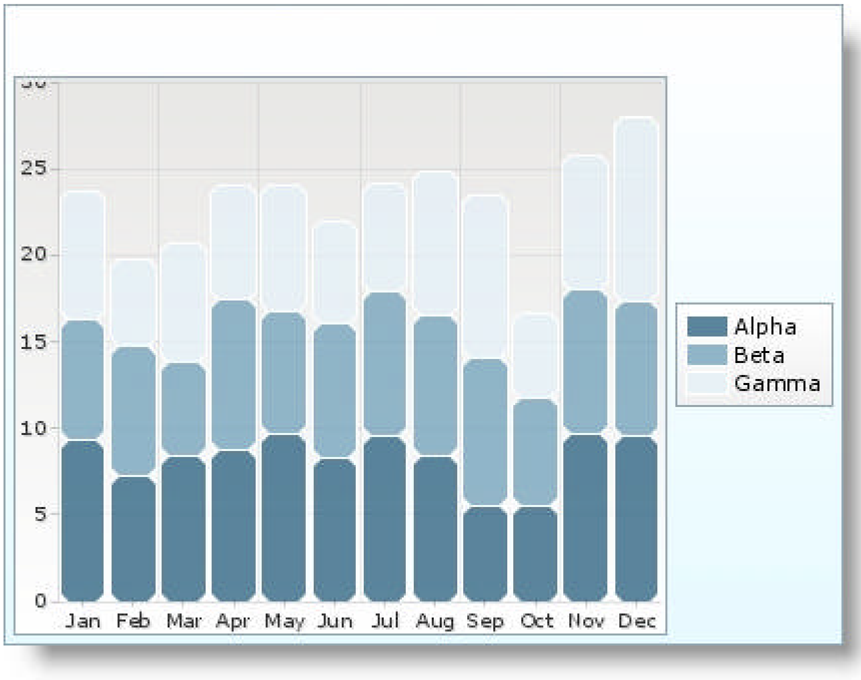
Set Group Type of a Chart (Section 4.5.3.5)

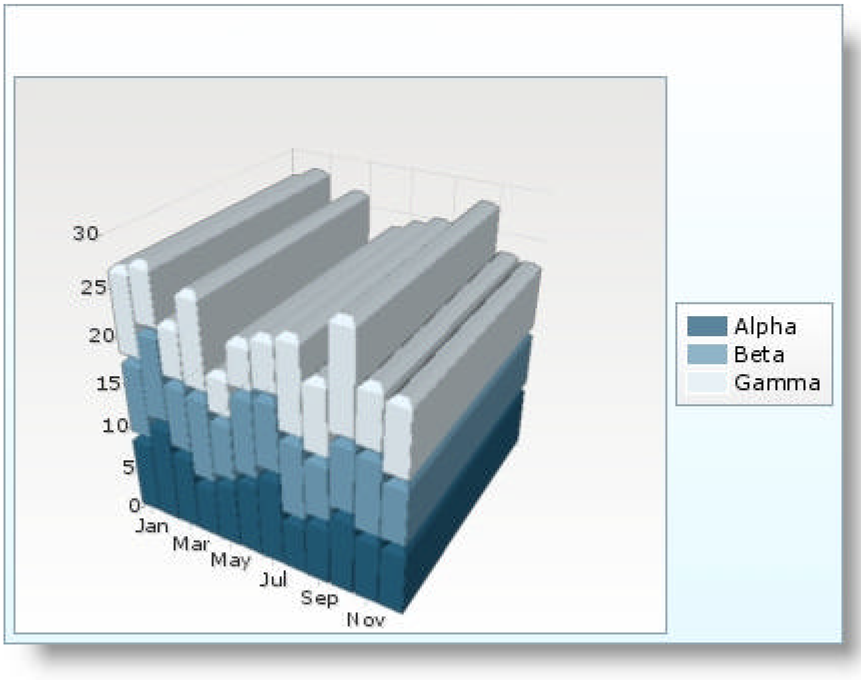
4.5.3.2 Stack Groups

Stacked groups show the relationship of individual items to the whole, comparing the contribution of each value to a total across categories or time.

To create a stacked chart, set the groupType attribute to "stack".

Below are examples of a stacked column chart in 2D and 3D.





Related Topic

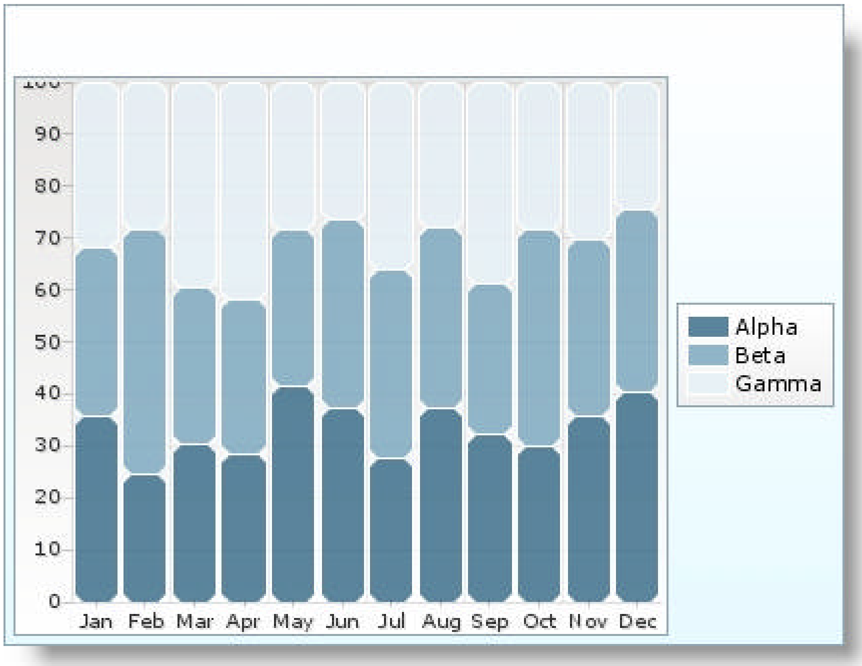
Set Group Type of a Chart (Section 4.5.3.5)

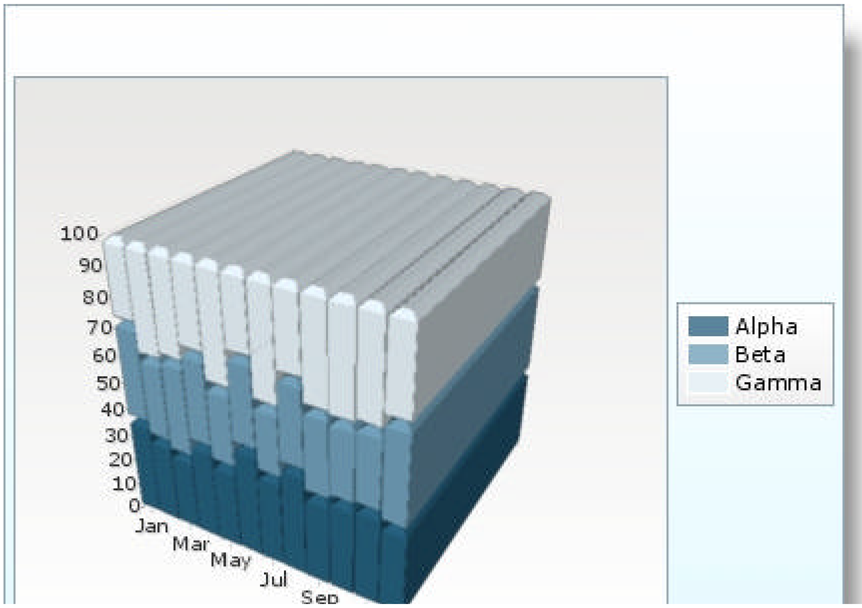
4.5.3.3 Stack 100 Groups

100% stacked column charts compare the percentage each value contributes to a total across categories or over time.

To create a 100% stacked chart, set the groupType attribute to "stack100".

Below are examples of a 100% stacked column chart in 2D and 3D.





Related Topic

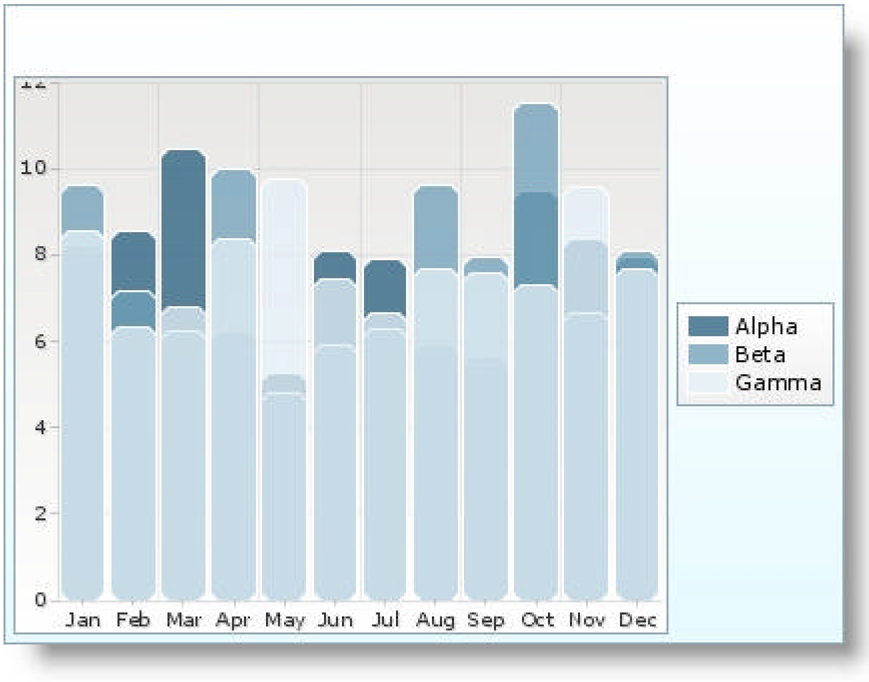
Set Group Type of a Chart (Section 4.5.3.5)

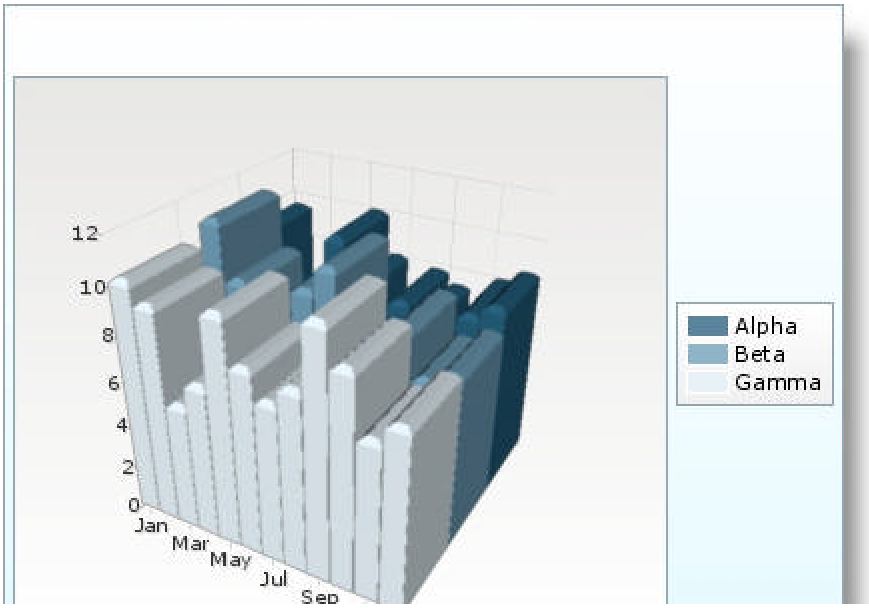
4.5.3.4 Overlay Groups

An overlay chart essentially shows each of the series as separate charts that are overlaid, resulting in a single charting window that has the same labels and categories.

To create an overlay chart, set the groupType attribute to "overlay".

Below are examples of an overlay column chart in 2D and 3D.





Related Topic

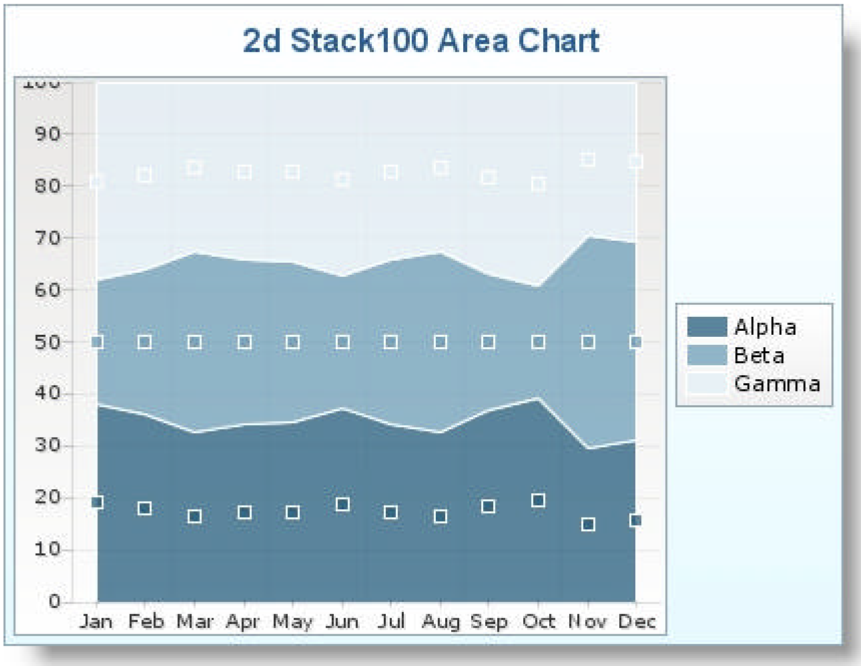
Set Group Type of a Chart (Section 4.5.3.5)

4.5.3.5 Set Group Type of the Chart

The data on a *WebChart*™ can be grouped into different types of arrangements to better visualize the relationship between the different data sets.

This topic will demonstrate how to set the group type of your chart:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



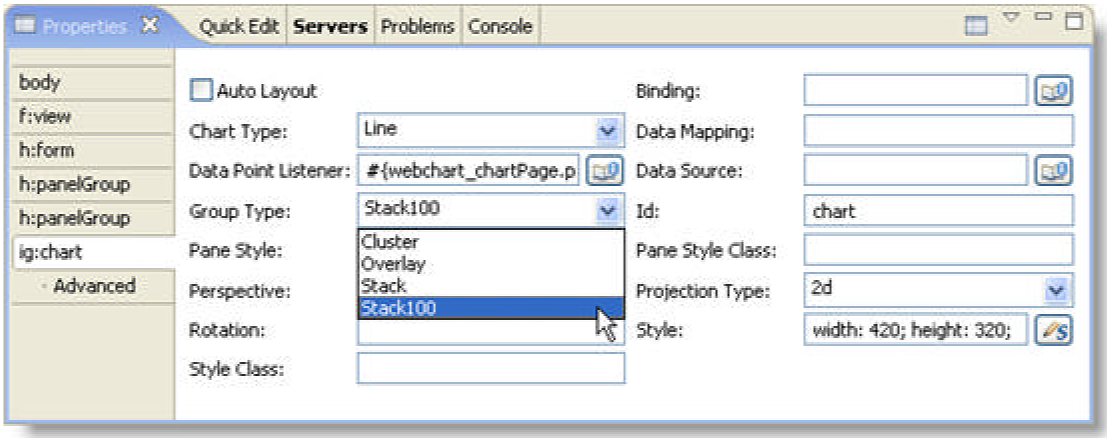
To set the group type using code:

1. Set the <groupType> attribute of the chart tag:
   In JSP:

```
<ig:chart
    id="chart"
    chartType="Area"
    groupType="Stack100"
    projectionType="2d"
    style="width: 420; height: 320"
    dataPointListener="#{webchart_chartPage.processDataPoint}">
```

To set the group type using RAD:

1. In Design view, select the chart component on your page.
2. In the Properties tab, select ig:chart.
3. In the Group Type drop down list, select the group type.



☐ Auto Layout

Chart Type: Line

Data Point Listener: #{webchart_chartPage.p

Group Type: Stack100

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 2d

Style: width: 420; height: 320;

4.5.4 Using WebChart

This section is your gateway to important conceptual and task-based information that will help you to use the various features and functionalities provided by the *WebChart*™ component.

- **Appearance (Section 4.5.4.1)**
- **Binding WebChart to Data (Section 4.5.4.2)**
- **Designing the Look and Feel (Section 4.5.4.3)**
- **Interaction with WebChart (Section 4.5.4.4)**
- **Working with Axes (Section 4.5.4.5)**
- **Working with Markers (Section 4.5.4.6)**
- **Working with the Legend (Section 4.5.4.7)**

4.5.4.1 Appearance

This section provides useful information on customizing the appearance of your charts. Click the links below to learn about combining series, WebCharts, and WebChart types, as well as how to style your chart.

- **Combining Series (Section 4.5.4.1.1)**
- **Combining WebCharts (Section 4.5.4.1.2)**
- **Combining WebChart Types (Section 4.5.4.1.3)**
- **Layout (Section 4.5.4.1.4)**
- **Object Shapes (Section 4.5.4.1.5)**
- **Styling Using Extended CSS (Section 4.5.4.1.6)**

4.5.4.1.1 Combining Series

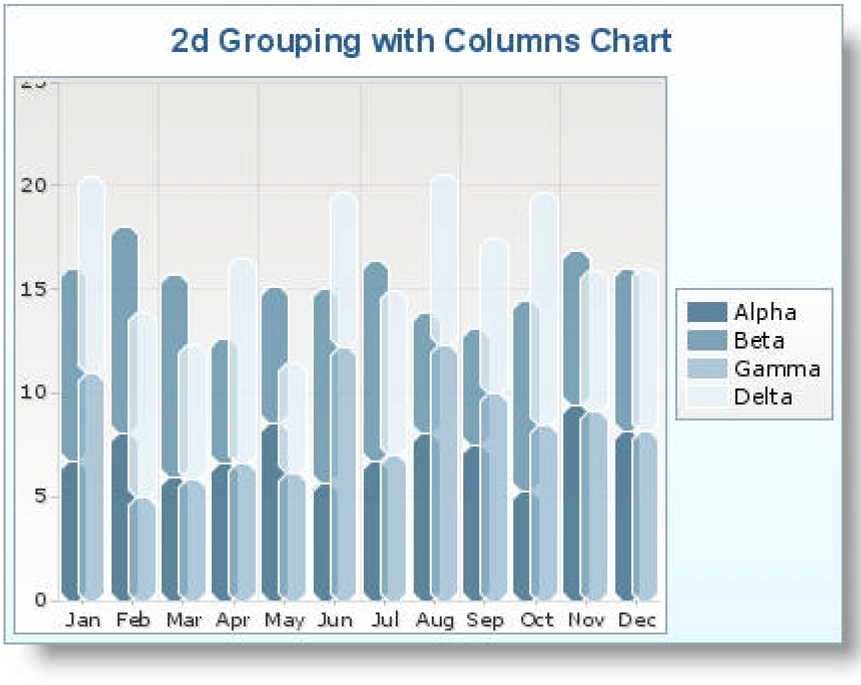
Combining series allow you to display different series groups in one chart. By nesting series groups with other series groups, you can create subgroups of series. To combine series, you simply define the groupType within the <ig:seriesGroup> tag.

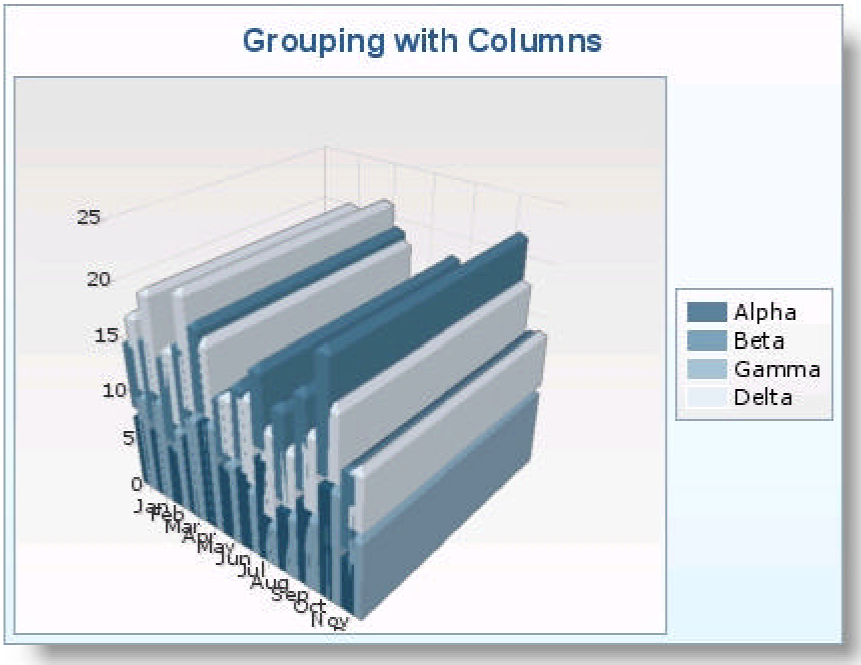
Each series within the <ig:seriesGroup/> tag are grouped according to the groupType attribute. If, however, there is not a groupType defined within the <seriesGroup/>, the groupType for the series will then be inherited from the above levels.

In JSP:

```
<ig:seriesGroup groupType="stack">
  .......
</ig:seriesGroup>
<ig:seriesGroup groupType="overlay">
  .......
</ig:seriesGroup>
```

The following screen shots show an example of grouping columns in 2D and 3D charts.




Grouping with Columns
25
20
15
10
5
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Alpha
Beta
Gamma
Delta

4.5.4.1.2 Combining WebCharts

The *WebChart*™ component allows you to display separate charts as one chart. This allows the user to see the data represented in different ways on one chart and compare the data easily and quickly.

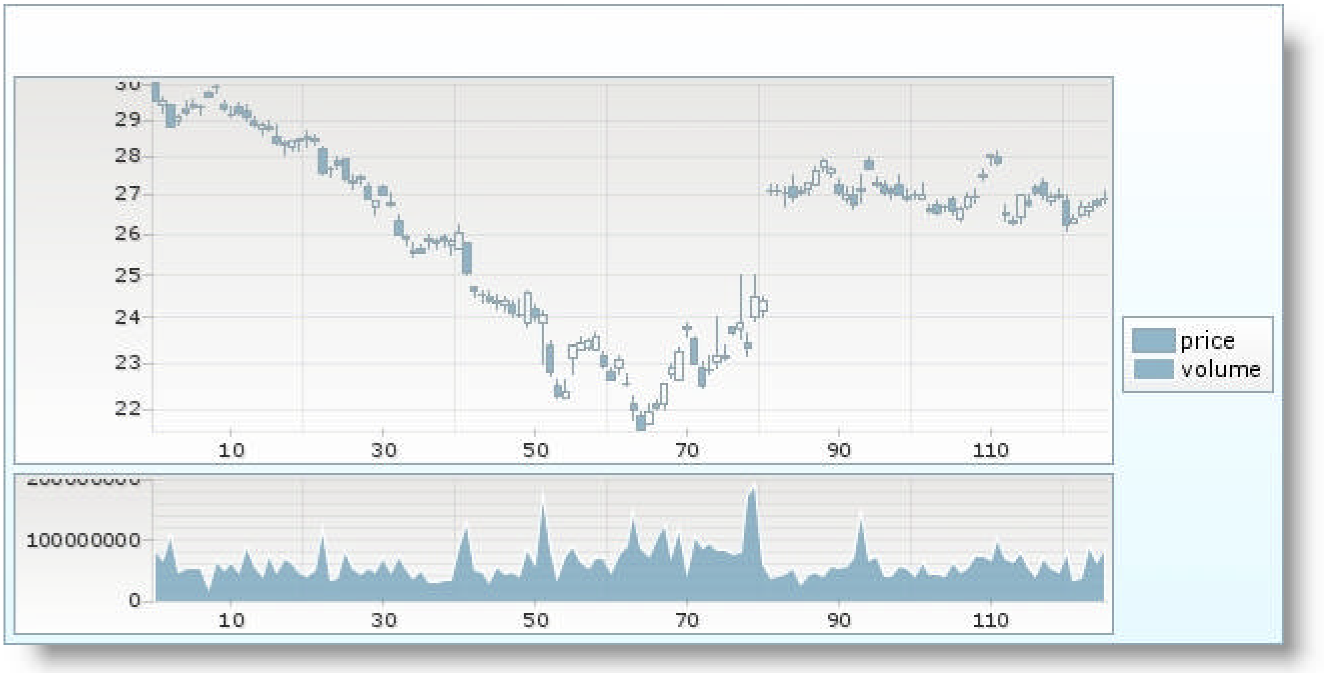
You can combine WebCharts by utilizing multiple <ig:chart> tags.

In the following example, both charts inherit the same x-axis. The candlestick chart has padding on the left so that the charts can be lined up.

In JSP:

```
<ig:axis type="x" autoTickMarks="true" autoGridLines="true" autoRange="true" />
<ig:chart chartType="candlestick"
      ......
</ig:chart>
<ig:chart chartType="area"
      ......
</ig:chart>
```

The screen shot below shows a combined chart displaying a candlestick chart and an area chart.



4.5.4.1.3 Combining WebChart Types

The *WebChart*™ component allows you to layer different charts on top of each other with different data sets.

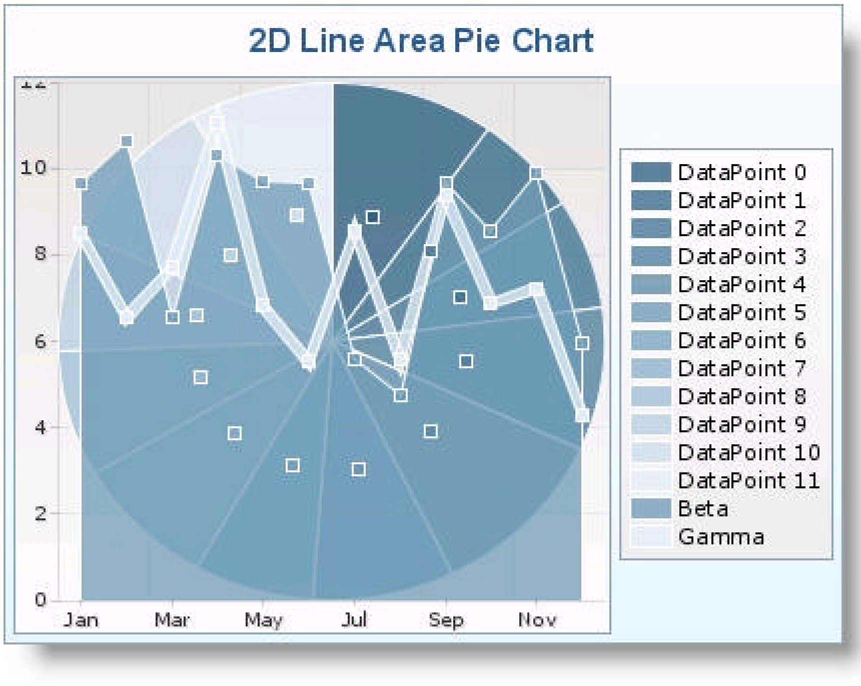
This can be used to show how a set of data can be compared to another set of data with different chart types.

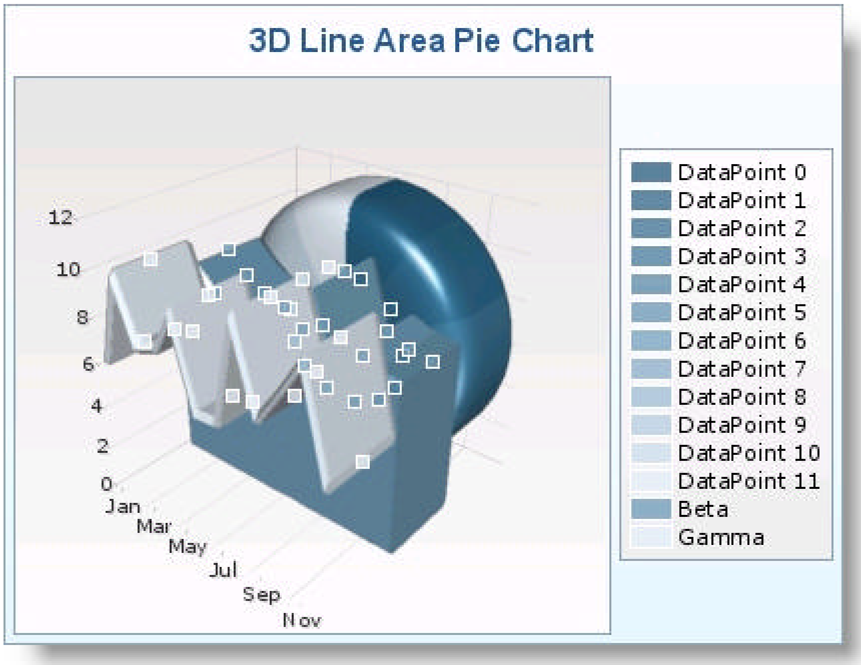
You can combine WebCharts by defining the nested chartType within the <ig:series> tag. The chartType defined in the <ig:chart> is the main chartType with the subsequent charts nested within it.

In JSP:

```
<ig:chart chartType="line">
...WebChart attributes for line chart
  <ig:series chartType="area" ...WebChart attributes for area chart... />
  <ig:series chartType="pie" ...WebChart attributes for pie chart... />
</ig:chart>
```

The following screen shots show a combination line/area/pie chart in 2D and 3D




3D Line Area Pie Chart
12
10
8
6
4
2
0
Jan
Mar
May
Jul
Sep
Nov
DataPoint 0
DataPoint 1
DataPoint 2
DataPoint 3
DataPoint 4
DataPoint 5
DataPoint 6
DataPoint 7
DataPoint 8
DataPoint 9
DataPoint 10
DataPoint 11
Beta
Gamma

4.5.4.1.4 Layout

This section discusses the layout of the legend and caption. They can be positioned by setting the position attribute. The possible values are:

- bottom
- left
- right
- top

In JSP:

```
<ig:caption position="bottom"... />
```

4.5.4.1.5 Object Shapes

Webchart™ allows you to change the look and feel of the standard series in charts such as column, pie and doughnut.

To find out more about object shapes in different charts, click the following links.

- **Column shapes (Section 4.5.4.1.5.1)**
- **Pie and doughnut shapes (Section 4.5.4.1.5.2)**

4.5.4.1.5.1 Column Shapes

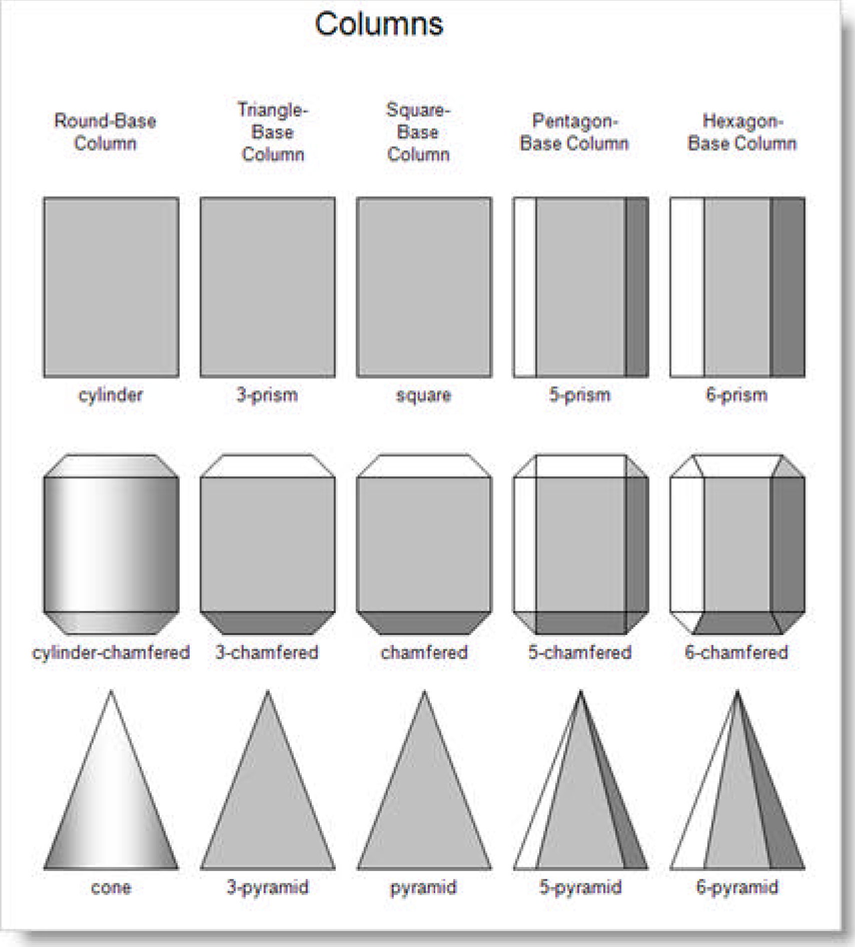
Each series within the *WebChart*™ component can be displayed as different shapes. This helps the user to differentiate easily and quickly between the different series. A shapeType attribute is defined within the <ig:series> tag.

In JSP:

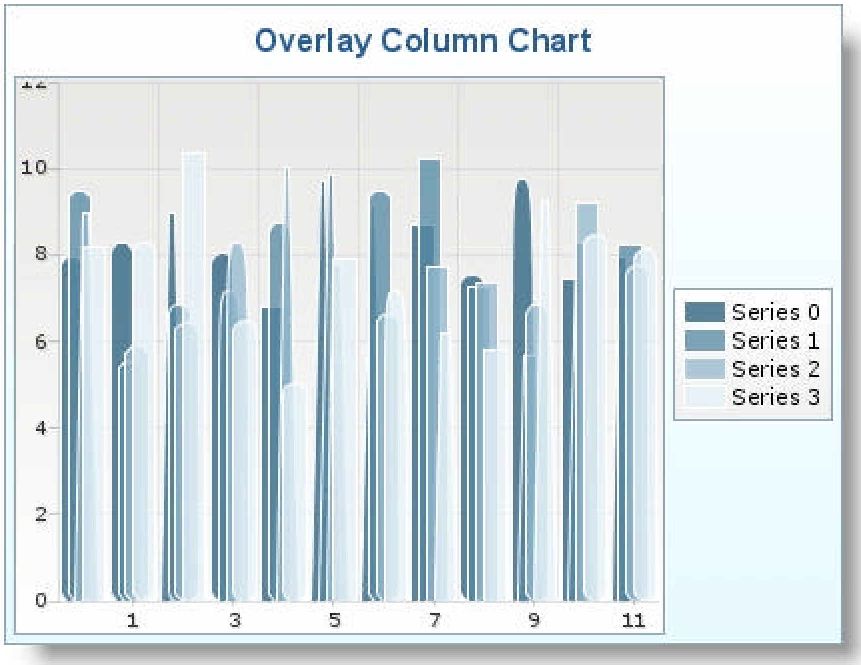
```
<ig:series shapeType="square" ..." />
```

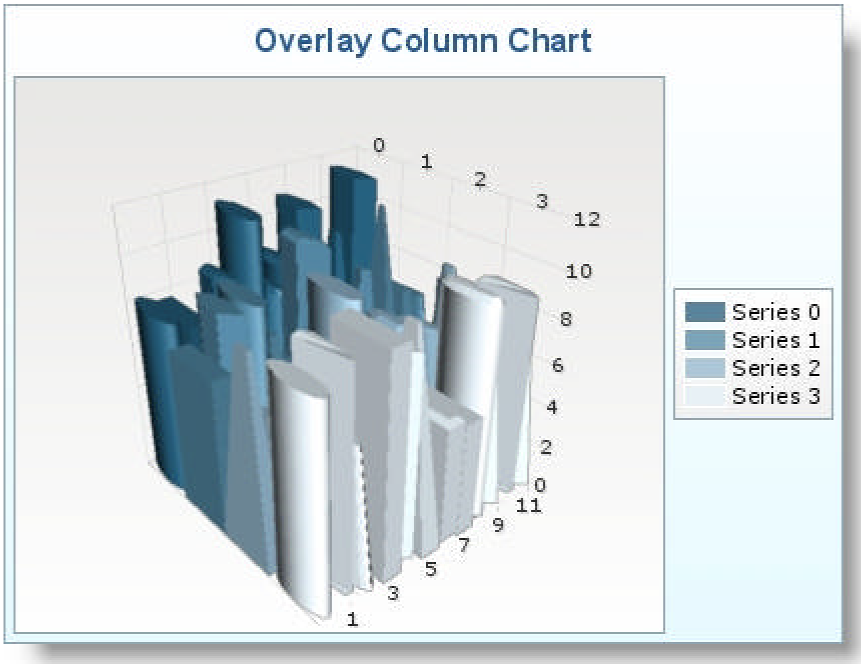
The table below describes the different column shape values.

Note: All shapes in each row of the table produce identical-looking columns in 2D charts.



The following screenshots show column shapes for both 2D and 3D charts.


Overlay Column Chart
12
10
8
6
4
2
0
1
3
5
7
9
11
Series 0
Series 1
Series 2
Series 3


Overlay Column Chart
0
1
2
3
12
10
8
6
4
2
0
11
9
7
5
3
1
Series 0
Series 1
Series 2
Series 3

4.5.4.1.5.2 Pie and Doughnut Shapes

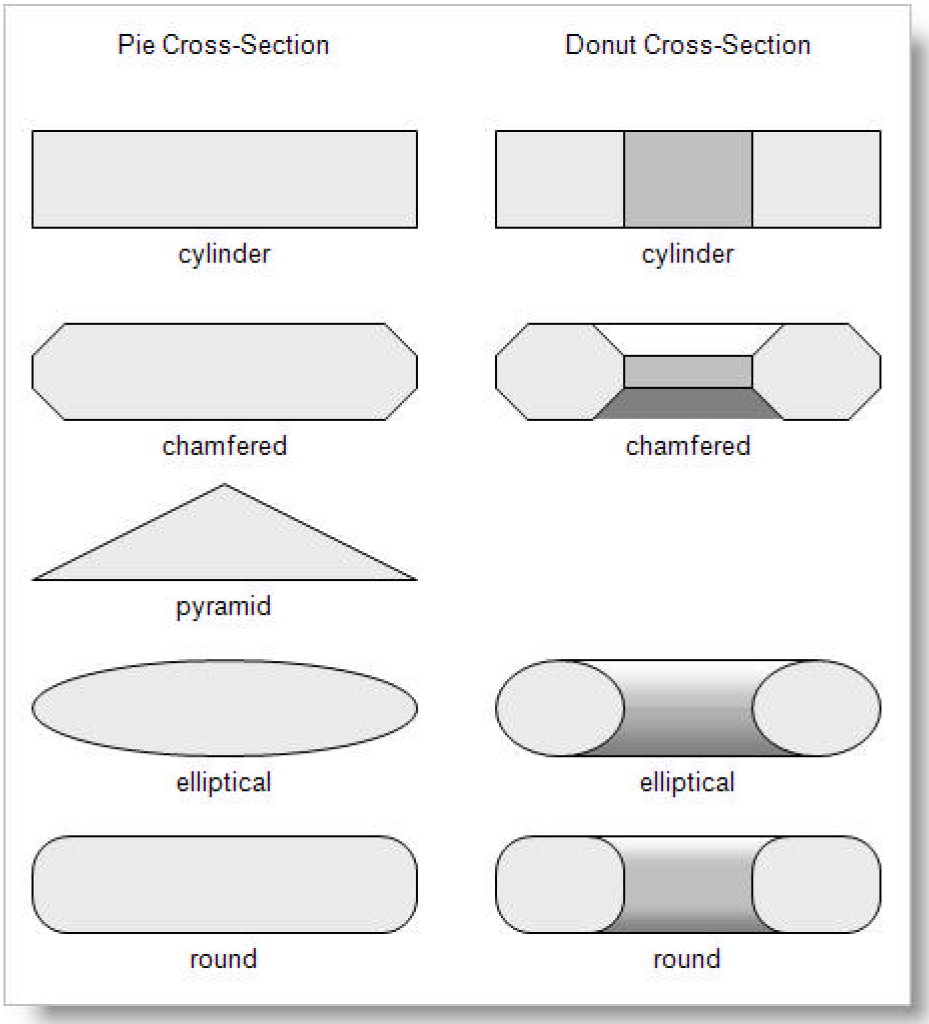
The *WebChart*™ component allows you to change the circular shape of pie and doughnut charts. This is done to change the look and feel of the standard pie and doughnut chart. A shapeType attribute is defined within the <ig:series/> tag.

The shapeType for pie and doughnut charts specifies the cross-section which is visible only in 3D charts.

In JSP:

```
<ig:series shapeType="chamfered" ..." />
```

The five pie and doughnut shape values are listed in the table below.



4.5.4.1.6 Styling Using Extended CSS

To allow specification of more complete styling attributes, a number of upward-compatible extensions have been made to the standard CSS syntax.

Background Gradients

Extended CSS supports the background-gradient-color and background-gradient-type attributes, which can also be used within the standard background shortcut to specify a gradient background. Transparent gradients are fully supported.

The background gradient runs from the background-color to the background-gradient-color in the direction specified by background-gradient-type, as shown below for a background color of black and a background gradient color of white.

The following example shows the specification of a horizontal white-to-black gradient using Extended CSS in the full form:

```
style="background-color: white; background-gradient-color: black;
       background-gradient-type: horizontal"
```

The following example shows specification of the same gradient using the background attribute shortcut:

```
style="background: white black horizontal"
```

Drop Shadow

Extended CSS supports the shadow-color attribute, used to add a drop shadow to displayed elements.

The following example shows the specification of a white text with a black drop shadow.

```
style="color: white; shadow-color: black"
```

4.5.4.2 Binding WebChart to Data

Binding is implemented at the data point level. The JSP files declare a series and then bind the contents of a series. The data source is defined in the backing bean.

The attributes that need to be set are:

- dataMapping -- Specifies the get() method on the objects of the collection to use to retrieve the individual data points.
- dataSource -- Specifies the collection that contains the data for the chart.

In JSP:

```
<ig:chart chartType="column">
    <ig:series dataMapping="value: europe" dataSource="#{chartbean.dataSource}"/>
    <ig:series dataMapping="value: asia" dataSource="#{chartbean.dataSource}"/>
    <ig:series dataMapping="value: us" dataSource="#{chartbean.dataSource}"/>
</ig:chart>
```

If the data mapping is merged, the data model value overrides it. So in the example above, the dataSource attribute could have been defined within the containing chart.

Binding to a Common Data Model

The following code snippet shows an example where it is assumed that all series share a common data model, and that all data points retrieve their "x" value from the "distance" entry:

In JSP:

```
<ig:chart chartType="column" dataMapping="x: distance">
    <ig:series dataMapping="value: column1"
                  dataSource="#{webchart_numbersSource.sparse1}"/>
    <ig:series dataMapping="value: column2"
                  dataSource="#{webchart_numbersSource.sparse2}"/>
    <ig:series dataMapping="value: column3"
                  dataSource="#{webchart_numbersSource.sparse3}"/>
</ig:chart>
```

4.5.4.3 Designing the Look and Feel

Click the following links to view topics written specifically to help you design the look and feel of the *WebChart* component.

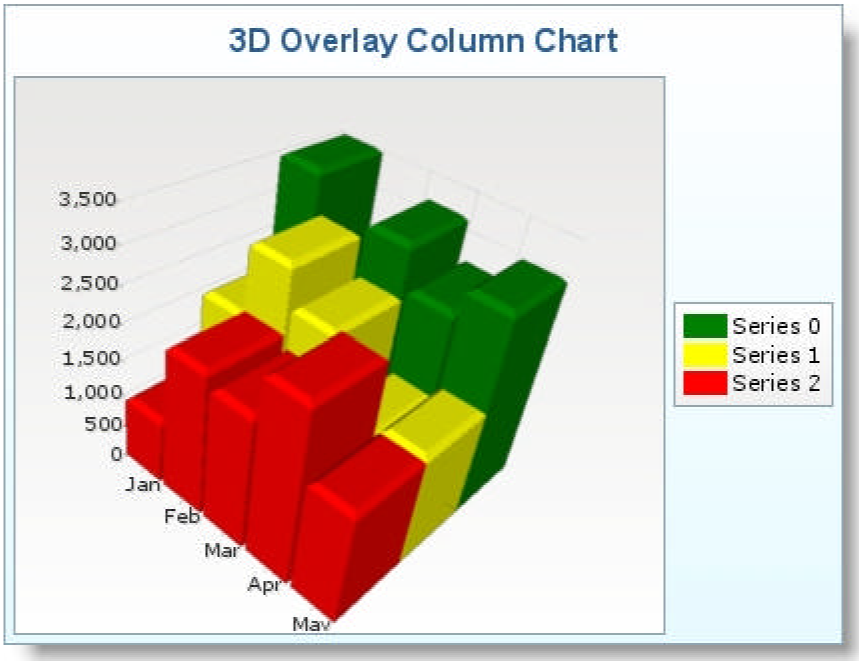
- **Customize Column Charts (Section 4.5.4.3.1)**
- **Customize Pie Charts (Section 4.5.4.3.2)**
- **Resize 3D Pie Charts (Section 4.5.4.3.3)**

4.5.4.3.1 Customize Column Charts

The WebChart component supports styling, which helps you to provide a consistent look and feel for your applications that use the WebChart component. For more information see **Themes, Styles and Style Classes (Section 5.3)**. However, you may want to override the styling classes and use your own colors for the series on your WebChart. This topic explains how to customize the colors on your column chart.

> **Note:** This topic assumes that you already created a column chart. For information on how to create a column chart, see **Column Chart (Section 4.5.2.5)**.

When you save and run your project after completing the following steps, your chart should look similar to the chart below.



To customize the look of a series in a column chart:

1. In the igf_chart.css file, located at INSTALL_DIRECTORY\resources\infragistics\themes\THEME_COLOR, create color classes for the series in your column chart , as shown in the code below:

   > **Note:** THEME_COLOR is the theme specified in your web.xml file.

   In CSS:

```
.red {
  background: red;
}

.green {
  background: green;
}

.yellow {
  background: yellow;
}
```

2. Within the <ig:series> tag of the <ig:chart> tag, in your JSP file, set the dataStyleClass attribute to the color classes you specified in the igf_chart.css file, as shown in the JSP code below:

   In JSP:

   ```
   <ig:series dataStyleclass="green"
       dataMapping="value: population" dataSource="#{chartData.asiaPopList}"/>
   <ig:series dataStyleclass="yellow"
       dataMapping="value: population" dataSource="#{chartData.africaPopList}"/>
   <ig:series dataStyleclass="red"
       dataMapping="value: population" dataSource="#{chartData.laCarPopList}"/>
   ```

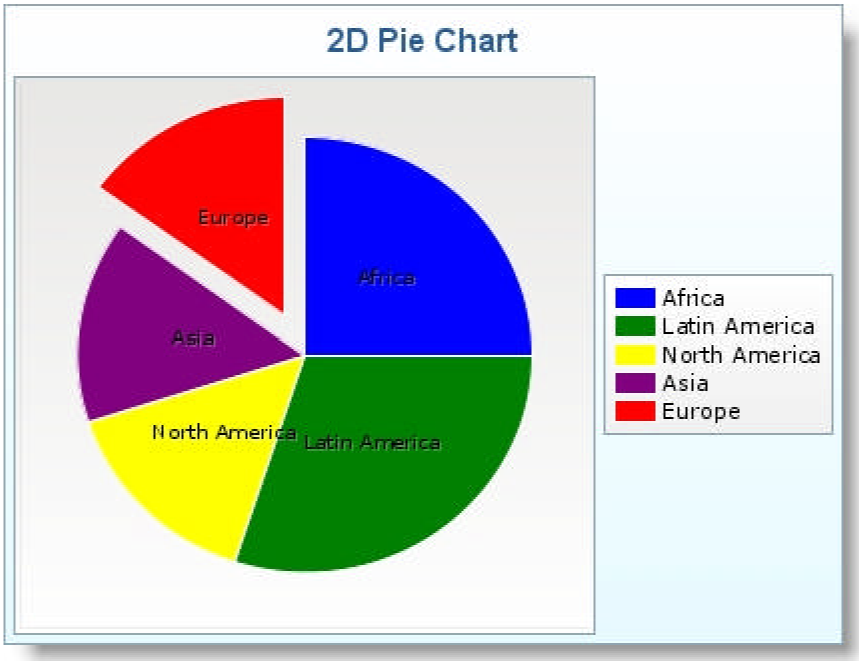
3. Save and run your project.

4.5.4.3.2 Customize Pie Charts

The WebChart component supports styling, which helps you to provide a consistent look and feel for your applications that use the WebChart component. For more information see **Themes, Styles and Style Classes (Section 5.3)**. However, you may want to override the styling classes and use your own colors for the series on your WebChart. This topic explains how to customize the colors on your pie chart.

> **Note:** This topic assumes that you already created a pie chart. For information on how to create a pie chart, see **Pie Chart (Section 4.5.2.9)**.

When you save and run your project after completing the following steps, your chart should look similar to the chart below.



To customize the look of a series in a pie chart:

1. In the igf_chart.css file, located at INSTALL_DIRECTORY\resources\infragistics\themes\THEME_COLOR, create color classes for the series in your column chart , as shown in the code below:

   > **Note:** THEME_COLOR is the theme specified in your web.xml file.

   In CSS:

```
.red {
    color: black; shadow-color: #40ffffff;
    font-family: verdana; font-size: 10;
    border: 1 solid white; border-join-style: join-miter;
    border-cap-style: cap-round;
    background: red;
}

.green {
    color: black; shadow-color: #40ffffff;
    font-family: verdana; font-size: 10;
    border: 1 solid white; border-join-style: join-miter;
    border-cap-style: cap-round;
    background: green;
}

.yellow {
    color: black; shadow-color: #40ffffff;
    font-family: verdana; font-size: 10;
    border: 1 solid white; border-join-style: join-miter;
    border-cap-style: cap-round;
    background: yellow;
}

.purple {
    color: black; shadow-color: #40ffffff;
    font-family: verdana; font-size: 10;
    border: 1 solid white; border-join-style: join-miter;
    border-cap-style: cap-round;
    background: purple;
}

.blue {
    color: black; shadow-color: #40ffffff;
    font-family: verdana; font-size: 10;
    border: 1 solid white; border-join-style: join-miter;
    border-cap-style: cap-round;
    background: blue;
}
```

2. In your backing bean, place the following code:

   In Java:

```
public static String[] chartColors =
      new String[] {"blue", "green", "yellow", "purple", "red"};
public static int chartColorsCounter = 0;
public String getColors()
{
    if (country.equals("Africa") return chartColors[0];
    else if (country.equals("Latin America")) return chartColors[1];
    else if (country.equals("North America") return chartColors[2];
    else if (country.equals("Asia") return chartColors[3];
    else if (country.equals("Europe") return chartColors[4];
}
```

3. Within the <ig:series> tag in your JSP file, set the dataStyleClass attribute to the color classes of the igf_chart.css file, as shown in the JSP code below:

   In JSP:

```
<ig:series
    radius="80"
    dataMapping="dataStyleClass: colors;
    legendCaption: countryName;
    explodedRadius: explodedRadius;
```

```
        value: population2004;
        markerCaption: countryName;
        markerCaptionVisible: captionVisible" />
```

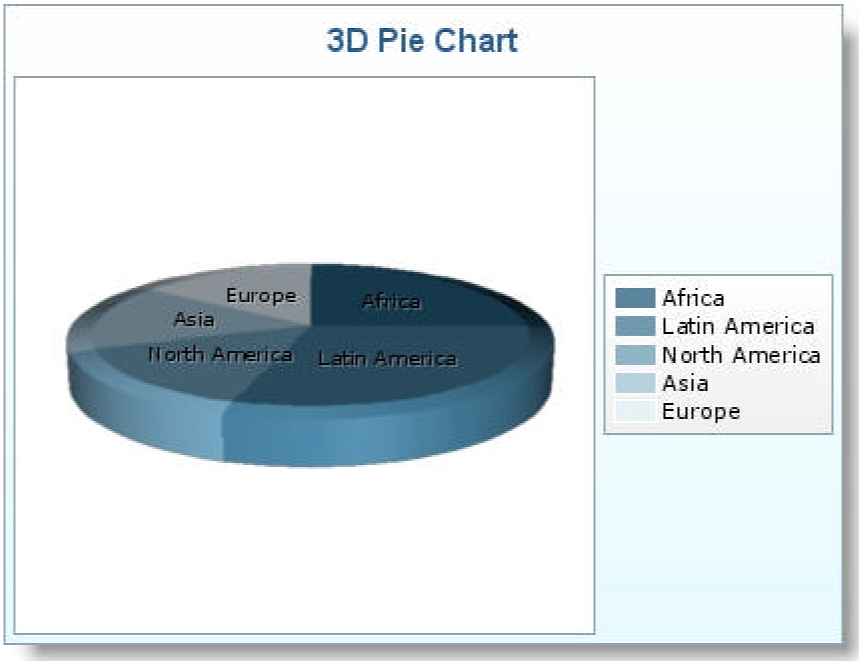
4. Save and run your project.

4.5.4.3.3 Resize 3D Pie Charts

You can resize your 3D pie chart using the z axis. This topic will help you to customize the size of a 3D pie chart.

> **Note:** This topic assumes that you already created a pie chart. For information on how to create a pie chart, see **Pie Chart (Section 4.5.2.9)**.

When you save and run your project after completing the following steps, your chart should look similar to the chart below.



To customize the size of a 3D pie chart:

1. Set the following attributes of the <ig:axis> tag in your JSP file:
   - type -- z
   - minimumValue -- "-5"
   - maximumValue -- "5"

   In JSP:

   ```
   <ig:axis type="z"
       minimumValue="-5"
       maximumValue="5">
   </ig:axis>
   ```

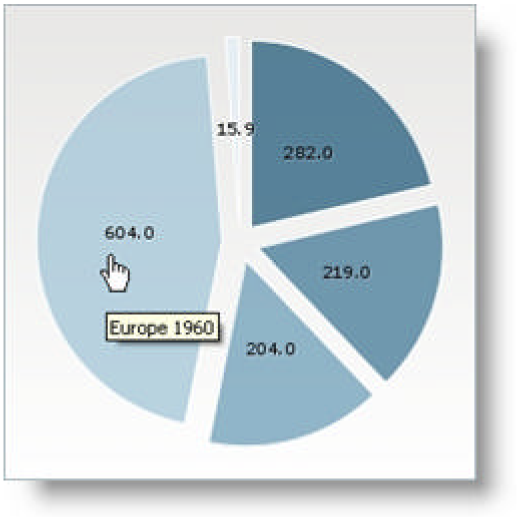
2. Save and run your project.

4.5.4.4 Interaction with WebChart

Your end user can interact with the *WebChart* component with the use of a ToolTip. You can set the ToolTip to a helpful message that is displayed to your end users when the hover over a data point in the chart. For example, if your end user scrolls of a data segment of a pie chart, then the ToolTip can appear as the value of the segment.

A ToolTip is attached to a data point or series in the same way as any other attribute.

The following example code demonstrates how to add a ToolTip to your chart.



How to Add a ToolTip to a Series

In JSP:

```
<ig:series tooltipCaption="ToolTip"
    markerBulletVisible="TRUE"
    markerCaptionVisible="TRUE"
    markerCaption="Label"
    dataSource="#{webchart_populationSource.population1960}"
    dataMapping="value: population;" />
```

How to Add a ToolTip to a Series Dynamically

In JSP:

```
<ig:series radius="80"
    dataSource="#{webchart_populationSource.population1960}"
    dataMapping="tooltipCaption: captionString;
      value: population;
      markerCaption: label;
      markerCaptionVisible: labelVisible; " />
```

In Java:

```
public String getCaptionString() {
  return population.toString();
}
```

4.5.4.5 Working with Axes

This section provides you with useful information about task-based procedures on working with chart axes.

- **About Axes (Section 4.5.4.5.1)**
- **Add an Axis (Section 4.5.4.5.2)**
- **Add Grid Lines to an Axis (Section 4.5.4.5.3)**
- **Add Tick Marks to an Axis (Section 4.5.4.5.4)**
- **Modify the Axis Range (Section 4.5.4.5.5)**

4.5.4.5.1 About Axes

An axis is a line which represents one dimension upon which chart data is plotted. The WebChart components all require at least two axes, except for the pie and doughnut chart which do not require axes. 3D charts require a z-axis to give depth to the chart.

An axis defined at the top level will be inherited by all the other tags. If an axis is defined within the lower tags, then it will override the inherited value.

An axis defines the scale for a series, as well as attributes for the visible axis and associated grid lines.

An axis object can define:

- A caption string which is rendered next to the axis whenever the axis is used.
- A set of marks which are drawn as tick marks whenever the axis is used.

Related Topics

Add an Axis (Section 4.5.4.5.2)

Add Grid Lines to an Axis (Section 4.5.4.5.3)

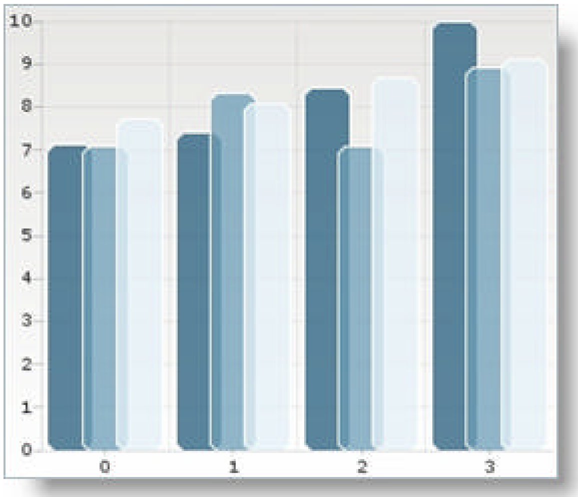
Add Tick Marks to an Axis (Section 4.5.4.5.4)

Modify the Axis Range (Section 4.5.4.5.5)

4.5.4.5.2 Add an Axis

The *WebChart*™ component allows you to add an axis to your chart to allow your end user to see the values quickly and easily. For example, if your webchart displayed sales for a year, by looking at the axis, your end user can easily see which month had the best sales.

The following code demonstrates how to add an axis to a WebChart.



In JSP:

```
<ig:axis type="x"
         autoRange = "true"
         autoTickMarks = "true"
         autoGridLines = "true" />
```

In Java:

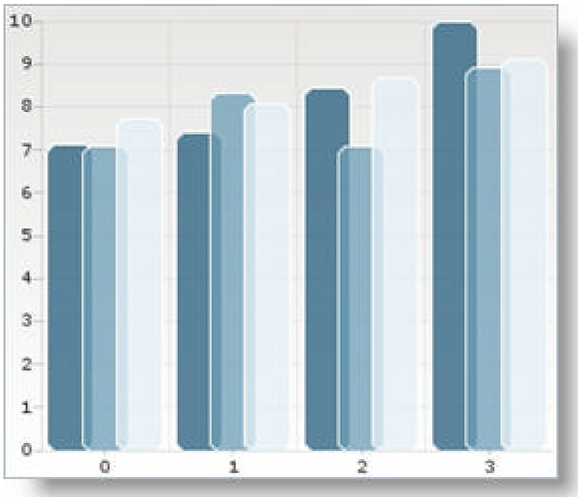
```
  Axis xAxis = new Axis();
  xAxis.setType("X");
  xAxis.setAutoRange(true);
  xAxis.setAutoTickMarks(true);
  xAxis.setAutoGridLines(true);
```

4.5.4.5.3 Add Grid Lines to an Axis

Grid lines are lines that are displayed perpendicular to an axis at various intervals. The Gridline property of an Axis object, control the grid lines that appear at regular intervals. You can set the grid lines to appear automatically by setting the autoGridLines attribute of the Axis object to True.

You can also manually set the gridLine attribute to display the grid lines at intervals you specify. Each gridLine attribute specifies one grid line on your chart.

The following example code demonstrates how to add grid lines to an axis.



In JSP:

```
<ig:axis type="x"
         minimumValue="0"
         maximumValue="10">
    <ig:gridLine value="1" range="10"/>
</ig:axis>
```

In Java:

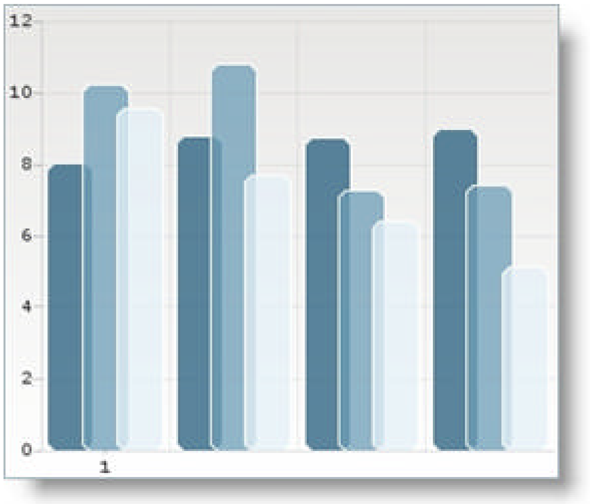
```
  GridLine gridLine = new GridLine();
  gridLine.setValue(1);
  gridLine.setRange(10);
```

4.5.4.5.4 Add Tick Marks to an Axis

Tick marks are small marks that are displayed along an axis at various intervals. The TickMark property of an Axis object controls the tick marks that appear at specified intervals. You can set the tick marks to appear automatically by setting the autoTickMarks attribute of the Axis object to True.

You can also manually set the tickMark attribute to display the tick marks at intervals you specify. Each tickMark attribute specifies one tick mark on your chart.

The following example code demonstrates how to add tick marks to an axis.



In JSP:

```
<ig:axis type="x">
    <ig:tickMark value="0" tickCaption="1"/>
</ig:axis>
```

In Java:

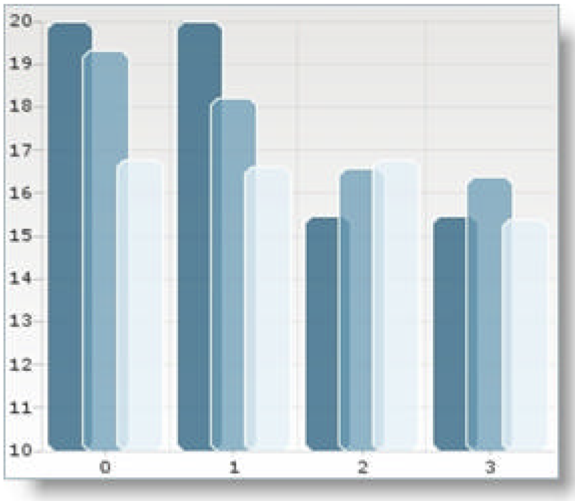
```
  TickMark tickMark = new TickMark();
  tickMark.setValue(0);
  tickMark.setTickCaption("1");
```

4.5.4.5.5 Modify the Axis Range

On a numeric axis, the axis range is the difference in numeric values from the beginning of the axis to the end. The range minimum is the value at the lowest point of the axis. The range maximum is the value at the highest point of the axis. By default, *WebChart*™ will use a minimum value of zero and calculate the maximum value for the axis range based on the data points. The automatic calculation of an axis' minimum and maximum values may not be appropriate for your set of data points.

For example, if your data points have a minimum value of 525, you may want to set the minimum value of the axis to 500 instead of zero. If you want manually set the minimum and maximum values of the axis, you can set the Axis object's autoRange property to False. In addition, you will also have to set the minimumValue and maximumValue properties.

The example code demonstrates how to modify the axis range.



In JSP:

```
<ig:axis type="x"
         autoRange="false"
         minimumValue="10"
         maximumValue="20">
</ig:axis>
```

In Java:

```
Axis axis = new Axis();
axis.setType("X");
axis.setAutoRange(false);
axis.setMinimumValue(10);
axis.setMaximumValue(20);
```

4.5.4.6 Working with Markers

This section provides you with useful information about the Marker, as well as access to task-based procedures on how to customize the Marker.

- **About Markers (Section 4.5.4.6.1)**
- **Add a Marker (Section 4.5.4.6.2)**

4.5.4.6.1 About Markers

A Marker is a visual element that displays the value of a DataPoint on your *WebChart*. Adding Markers to your WebChart will help your end users immediately identify a DataPoint's value even if the value falls between MajorGridlines or MinorGridlines. The Markers are attributes of the Series property.

You can set the following attributes:

- **markerBulletSize (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies the size of the bullet that appears next to the marker in pixels.
- **markerBulletStyleClass (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies the style class for the bullet.
- **markerBulletType (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- You can set this attribute to change the default circular shape of a bullet. The values for this attribute are:
  - circle
  - iTriangle
  - square
  - star
  - triangle
- **markerBulletVisible (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies if the bullet is visible, default value is False.
- **markerCaption (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies the text for the marker
- **markerCaptionStyleClass (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies the style class for the marker
- **markerCaptionVisible (JavaDocFiles\com\infragistics\faces\chart\taglib\html\SeriesTag.html)** -- Specifies if the marker is visible, default value is False

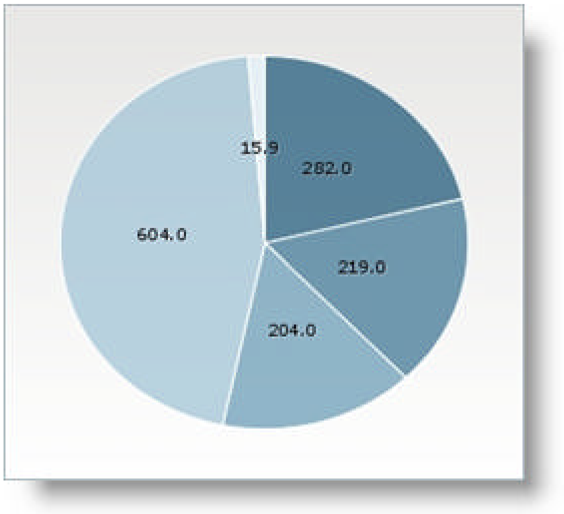
Related Topic

Add a Marker (Section 4.5.4.6.2)

4.5.4.6.2 Add a Marker

You can add a Marker to a Series or to an individual DataPoint. If you add a Marker to the Series, all DataPoints in that Series will use the Marker; however, if you also add a Marker to one of your DataPoints, that DataPoint will use its own Marker instead of the one you added to the Series. You can use this feature to draw attention to a particular DataPoint in your Series by giving the DataPoint its own Marker.

The following example code demonstrates how to add a Marker to a Series.



<u>How to Add a Marker to a Series</u>

In JSP:

```
<ig:series markerBulletVisible="TRUE"
    markerCaptionVisible="TRUE"
    markerCaption="Label"
    dataSource="#{webchart_populationSource.population1960}"
    dataMapping="value: population;" />
```

<u>How to Add a Marker to a Series Dynamically</u>

In JSP:

```
<ig:series radius="80"
    dataSource="#{webchart_populationSource.population1960}"
    dataMapping="value: population;
      markerCaption: label;
      markerCaptionVisible: labelVisible; " />
```

In Java:

```
public String getLabel() {
  return population.toString();
}

public Boolean getLabelVisible() {
  return Boolean.TRUE;
}
```

Related Topic

About Markers (Section 4.5.4.6.1)

4.5.4.7 Working with the Legend

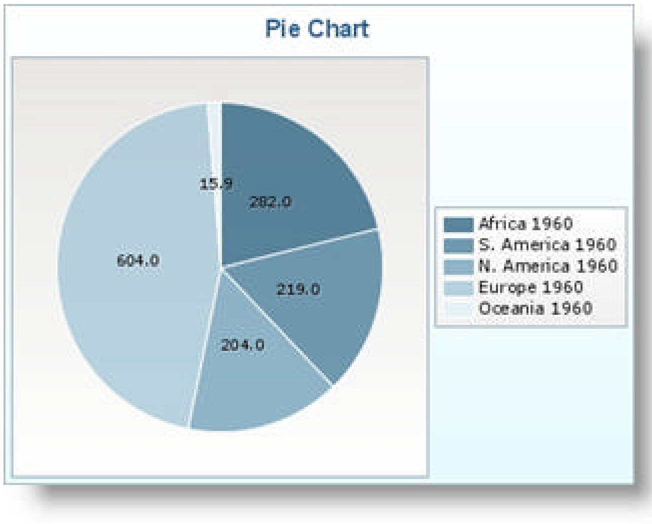
This section provides you with useful information about the Legend, as well as access to task-based procedures on how to customize the Legend.

- **Add the Legend (Section 4.5.4.7.1)**
- **Position the Legend (Section 4.5.4.7.2)**

4.5.4.7.1 Add the Legend

A legend displays information related to the data displayed on the chart to your end user. The *WebChart* control creates a Legend based on the Series or DataPoints in your chart. To make the Legend more user friendly, you can customize the Legend caption to specify the data types in the series. You can do this by setting the dataMapping attribute of the Series tag.

The following example code demonstrates how to add the Legend.



In JSP:

```
<ig:legend autoItems = "TRUE" />
<ig:series radius="80"
    dataSource="#{webchart_populationSource.population1960}"
    dataMapping="value: population;
      markerCaption: label;
      markerCaptionVisible: labelVisible;
      legendCaption: legendString" />
```

In Java:

```
//Creates the Legend
Legend legend = new Legend();
legend.setAutoItems(true);

//Returns the markerCaption for each data point in the series
public String getMarkerCaption() {
  return markerCaption;
}

//Returns if marker is visible
public String getLabelVisible() {
  return labelVisible;
}

//Returns the Legend caption for each data point in the series
public String getLegendString() {
  return legendString;
}
```

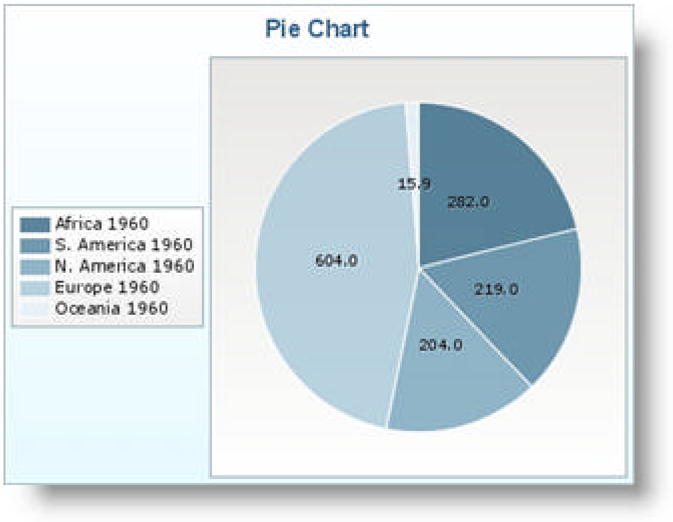
Related Topic

Position the Legend (Section 4.5.4.7.2)

4.5.4.7.2 Position the Legend

The *WebChart* component automatically places the Legend on the right-hand side of the chart; however, you can also position the Legend manually.

The following example code demonstrates how to position the Legend.



In JSP:

```
<ig:legend autoItems = "TRUE"
    position = "left" />
```

In Java:

```
//Creates the Legend
Legend legend = new Legend();
legend.setAutoItems(true);
legend.setPosition("Left");
```

Related Topic

Add the Legend (Section 4.5.4.7.1)

4.5.5 WebChart Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.6 WebDialogWindow

If you're looking for an area that houses all of the overview/task-based information related to the *WebDialogWindow*™ component, you've come to the right place. This section contains valuable information about WebDialogWindow, ranging from what the component does and why you would want to use it in your application, to step-by-step procedures on how to accomplish a common task using the component.

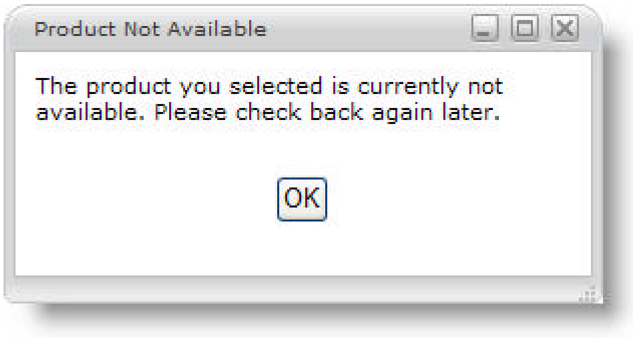
Click the links below to access important information about the WebDialogWindow component.

- **About WebDialogWindow (Section 4.6.1)** -- In this section you'll find any information that will help you to better understand the functionalities of the WebDialogWindow component, as well as why you would want to use it to as part of building your applications.
- **Getting Started with WebDialogWindow (Section 4.6.2)** -- In order to get you up and running as quickly as possible with the WebDialogWindow component, we've provided you with information on how to get started using the component (i.e., adding WebDialogWindow to a form).
- **Using WebDialogWindow (Section 4.6.3)** -- This section is your gateway to important task-based information that will help you to effectively use the various features and functionalities provided by the WebDialogWindow component.
- **WebDialogWindow Accessibility Compliance (Section 4.6.4)** -- This topic outlines the way in which WebDialogWindow meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebDialogWindow component.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.6.1 About WebDialogWindow

The WebDialogWindow™ control is able to bypass several pop-up blockers in modern browser by displaying as a simple <DIV> section in your web page, but appearing as a pop-up window. WebDialogWindow allows you to easily add dialog box functionality to any web application. The WebDialogWindow's content area, being a <DIV> itself, can contain any HTML or ASP.NET object. You can display WebDialogWindow as either as either a modal or non-modal dialog box.

With CSS based properties, you can manually customize the WebDialogWindow control by leveraging your existing style sheets.


Product Not Available

The product you selected is currently not available. Please check back again later.

WebDialogWindow supports the following features:

Modal/Non-Modal Modes - WebDialogWindow allows you to set whether the control displays as modal or non-modal. Choosing the right mode can control the flow of your application and guarantee an action from your end user when you need one.

Header - WebDialogWindow displays a header area, which can contain text as well as several buttons including close, minimize, and maximize to control the dialog window.

Resizing - When necessary, your end user can resize WebDialogWindow at run time.

Location - You can either manually set WebDialogWindow's start location or allow it to display automatically in the center of your application.

Window State - Your end user can minimize or maximize WebDialogWindows, or you can hide it from your end user altogether.

Note: JavaScript methods for the WebDialogWindow that begin with an underscore (_) are private methods. We recommend that you do not use these private methods in your web applications as they are not supported.

4.6.2 Getting Started with WebDialogWindow

Before You Begin

The following steps will show you how to get started with *WebDialogWindow™*. You will complete a series of steps in order to become familiar with the control's object model.

What You Will Accomplish

When you have completed the following steps, you will have successfully created your first application that leverages the WebDialogWindow control.

Follow these Steps:

1. Open the <ig:dialogWindow> tag and set the following attributes, as shown in the JSP code below:
   - style -- width:300px; height:150px
   - id -- htmlDialogWindowRender1
   - styleClass -- dialogWindow

   In JSP:

```
<ig:dialogWindow style="width:300px;height:150px"
                id="htmlDialogWindowRenderer1"
                styleClass="dialogWindow">
```

2. Set the header of your dialog window, as well as the minimize, maximize and close buttons. The following code demonstrates how to set the dialog windows caption and center it in the header area.
   In JSP:

```
<ig:dwHeader captionText ="Product Not Available">
    <ig:dwMinimizeBox></ig:dwMinimizeBox>
    <ig:dwMaximizeBox></ig:dwMaximizeBox>
    <ig:dwCloseBox></ig:dwCloseBox>
</ig:dwHeader>
```

3. Add content to your dialog window, as shown in the code below.
   In JSP:

```
<ig:dwContentPane>
   <f:verbatim escape="false">
      The product you selected is currently not available.
      Please check back again later.
      <br><br><br>
      <div align="center">
         <button>OK</button>
      </div>
   </f:verbatim>
</ig:dwContentPane>
```

4. Allow your end users to resize your dialog window by setting the resizer tag as shown in the code below.

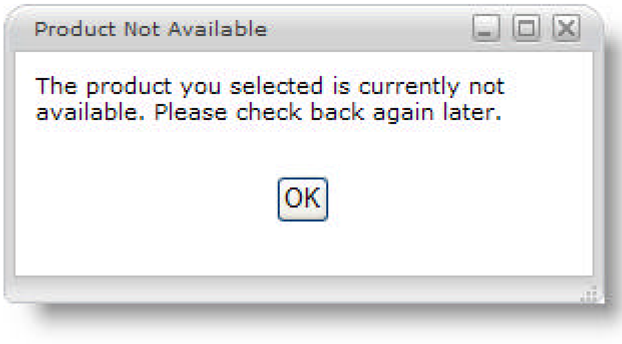
In JSP:

```
<ig:dwResizer></ig:dwResizer>
```

5. Close the <ig:dialogWindow> tag, as shown in the JSP code below.
   In JSP:

```
</ig:dialogWindow>
```

6. Run the project. It should look similar to the dialog window below.



Product Not Available

The product you selected is currently not available. Please check back again later.

4.6.3 Using WebDialogWindow

This section is your gateway to important task-based information that will help you to effectively use the various features and functionalities provided by WebDialogWindow™ control.

- **Add a Caption to the Dialog Window's Header (Section 4.6.3.1)**
- **Allow the End User to Resize the Dialog Window (Section 4.6.3.2)**
- **Center the Dialog Window (Section 4.6.3.3)**
- **Display the Dialog Window (Section 4.6.3.4)**
- **Maintain the Header's Width on Minimize (Section 4.6.3.5)**
- **Make the Dialog Window Modal (Section 4.6.3.6)**
- **Prevent the End User from Moving the Dialog Window (Section 4.6.3.7)**
- **Show the Minimize and Maximize Buttons (Section 4.6.3.8)**
- **Using WebDialogWindow with DateChooser (Section 4.6.3.9)**

4.6.3.1 Add a Caption to the Dialog Window's Header

Every window, no matter how insignificant, needs some sort of descriptive text in the header area. This way, end users will know, at-a-glance, what they are about to read in the window. The dialog window is no exception, and exposes useful attributes within the <ig:dwHeader> tag to accommodate a caption in the header area:

CaptionText – this attribute allows you to supply a descriptive string which is displayed in the header. The captions location depends on the CaptionAlignment property.

CaptionAlignment – this attribute allows you to align the caption to the left, right, or center.

The following code demonstrates how to set the dialog windows caption and center it in the header area.

In JSP:

```
<ig:dwHeader captionText="Product Not Available"
             captionAlignment="center">
```

4.6.3.2 Allow the End User to Resize the Dialog Window

If you are using the dialog window as a simple prompt dialog box with a single OK button, your end users may not need to resize it. However, if you are displaying a large amount of information, or even an entire web page in the dialog window, the end user may need more room to comfortably view all of the dialog window's contents. If you set the resizer attribute, the end user will be able to resize the dialog window to as large or small as they need it to comfortably view their data. Enabling the resizer displays a small resizing image in the lower, right corner of the dialog window. End users can grab this resizer and drag the dialog window until it is the right size for their taste.

The following code demonstrates how to enable the resizer.

In JSP:

```
<ig:dwResizer"></ig:dwResizer>
```

4.6.3.3 Center the Dialog Window

When you first place WebDialogWindow™ on your page, WebDialogWindow's initial position will default to center. If this location of the WebDialogWindow component is not appropriate for your web page's needs, you can always change the dialog window's initial location by setting the initialPosition attribute to manual and set the style attribute for the location of the dialog window.

The following code demonstrates how to manually display the dialog window on the web page.

In JSP:

```
<ig:dialogWindow initialPosition="manual" style="left:200px;top:100px">
```

In Java:

```
dialogWindow.setInitialPosition("manual");
dialogWindow.setStyle("left:200px; top:100px;");
```

4.6.3.4 Display the Dialog Window

By default, the WebDialogWindow™ component's windowState attribute is set to Normal, which means it will display on your web page when the page first loads. However, if you are using the dialog window as a popup dialog box, you will need to first hide the dialog window, and then display it through a button's Click event or some other means. Displaying the dialog window is as easy as hiding it, simply set the WindowState property to Normal.

A common scenario is when an end user clicks a button to postback some data to the server. If there is a problem with that data, you can display the dialog window as an error message along with a text box to allow the end user to correct the problem. In that button's Click event, all you need to do to display the dialog window is set WindowState to Normal.

The following code demonstrates how to display a dialog window inside a button's Click event.

1. Create a Javascript method to change the window's state to normal when a button is clicked.
   In Javascript:

   ```
   function igShowDialog() {
      var igJsDwNode = ig.dw.getDwJsNodeById("_id0:igDw001");
      if (igJsDwNode != null) {
         igJsDwNode.set_windowState(ig.dw.STATE_NORMAL);
      }
   }
   ```

2. Set the window's state to hidden when the page initially loads.
   In JSP:

   ```
   <ig:dialogWindow windowState="hidden" id="igDw001">
   ```

3. Create a button that calls the javascript method when it is clicked.
   In JSP:

   ```
   <input type="button" onclick="igShowDialog();return false;"
          value="Show Dialog" />
   ```

4.6.3.5 Maintain the Header's Width on Minimize

If you are allowing your end users to minimize the WebDialogWindow™ controls dialog window through the minimize box, the dialog window automatically hides the content pane and decreases the header's width to 20 pixels. If you would prefer to keep the header's width the same when it is minimized, then you will need to set the MinimizedWidth property off the Header object. An easy way to maintain the width of the header is to set the MinimizedWidth property to the same value as the dialog window's width.

The following code demonstrates how to set the header's width on minimize.

In JSP:

```
<ig:dialogHeader captionText="Dialog Caption" minimizedWidth="300">
```

4.6.3.6 Make the Dialog Window Modal

There may be times when you need input from your end users and cannot continue without it. In cases like these, you can display the WebDialogWindow™ control's dialog window as a modal dialog box. This is useful if an error occurs and you need your end user to correct the error before they can continue. Turning the dialog window into a modal dialog box requires you to set the modal attribute to True. Once the dialog window is modal, the end user will not be able to interact with any other part of the web page until the dialog window is dismissed again.

The following code demonstrates how to turn the dialog window into a modal dialog box.

In JSP:

```
<ig:dialogWindow modal="true">
```

In Java:

```
dialogWindow.setModal(true);
```

4.6.3.7 Prevent the End User from Moving the Dialog Window

If you allow your end user to have multiple dialog windows open on the web page at once, you may decide to give them the option to lock them in place. Your end users may appreciate this option, especially if they accidentally move a dialog window when they had it in the perfect location. This situation can arise if multiple dialog windows are displaying different customer statistics, such as a "card view" in a presentation grid. By setting the moveable attribute to False, the end user will not be able to move the dialog window around the web page. You can present this ability to your end user in a "Lock Windows in Place" option, perhaps. Keep in mind, if there are multiple dialog windows open at once, you'll have to set each moveable attribute individually.

The following code demonstrates how to lock a dialog window in place, preventing the end user from moving it.

In JSP:

```
<ig:dialogWindow movable="false">
```

In Java:

```
dialogWindow.setMovable(false);
```

4.6.3.8 Show the Minimize and Maximize Button

WebDialogWindow™ allows your end users to either minimize or maximize the dialog window by using two buttons in the header. Both found off the <ig:dwHeader> tag, the <ig:dwMaximizeBox> tag and the <ig:dwMinimizeBox> tag allow you to display both the maximize and minimize boxes, respectively. When the end user clicks the minimize box, the dialog window all but disappears, except for the header. The header itself decreases its width to a default size of 20 pixels. You can modify the minimized width manually; see **Maintain the Header's Width on Minimize (Section 4.6.3.5)** for more information. When the end user clicks the maximize box, the dialog window expands until it uses all of the available screen real estate within its parent page. The end user may wish to maximize the dialog window if there is a large amount of information in it, or if there is another web page displaying inside.

The following code demonstrates how to display both the maximize and minimize boxes in the dialog window's header.

In JSP:

```
<ig:dwHeader captionText="Dialog Header">
   <ig:dwMinimizeBox></ig:dwMinimizeBox>
   <ig:dwMaximizeBox></ig:dwMaximizeBox>
   <ig:dwCloseBox></ig:dwCloseBox>
</ig:dwHeader>
```

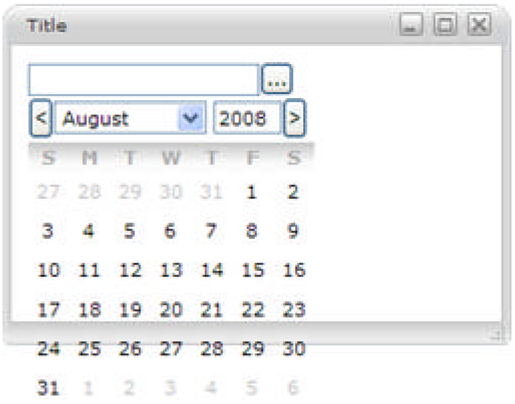
4.6.3.9 Using WebDialogWindow with DateChooser

When you create a modal WebDialogWindow that contains a DateChooser component, the calendar will appear behind the dialog window. This is due to a conflict with the z-index of DateChooser and WebDialogWindow.

A solution to this conflict is to set the z-index of DateChooser to be higher than 1000, which is the default value of the z-index of WebDialogWindow.

To resolve the conflict of the z-indexes, all you need to do is set the popStyleClass attribute of the <ig:dateChooser> tag to a style sheet where the z-index is set.

The following code demonstrates how to ensure DateChooser works correctly with WebDialogWindow.



Setting the z-index in the stylesheet:

```
.popStyle {
    z-index: 2000;
}
```

In JSP:

```
<ig:DateChooser id="dateChooserRenderer1"
    popupStyleClass="popStyle" />
```

4.6.4 WebDialogWindow Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.7 WebGrid

If you're looking for an area that houses all of the overview/task-based information related to the WebGrid component, you've come to the right place. This section contains valuable information about WebGrid, ranging from what the component does and why you would want to use it in your application, to step-by-step procedures on how to accomplish a common task using the component.

Click the links below to access important information about the WebGrid component.

- **About WebGrid (Section 4.7.1)** -- In this section you'll find any information that will help you to better understand the functionalities of the WebGrid component, as well as why you would want to use it to as part of building your applications.
- **Getting Started with WebGrid (Section 4.7.2)** -- In order to get you up and running as quickly as possible with the WebGrid control, we've provided you with information on how to get started using the component.
- **Using WebGrid (Section 4.7.3)** -- This section is your gateway to important task-based information that will help you to effectively use the various features and functionalities provided by the WebGrid component.
- **WebGrid Accessibility Compliance (Section 4.7.4)** -- This topic outlines the way in which WebGrid meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebGrid control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.7.1 About WebGrid

WebGrid is a highly functional, hierarchical, AJAX-enabled table that allows you to display your data in the style and manner that best fits your needs.

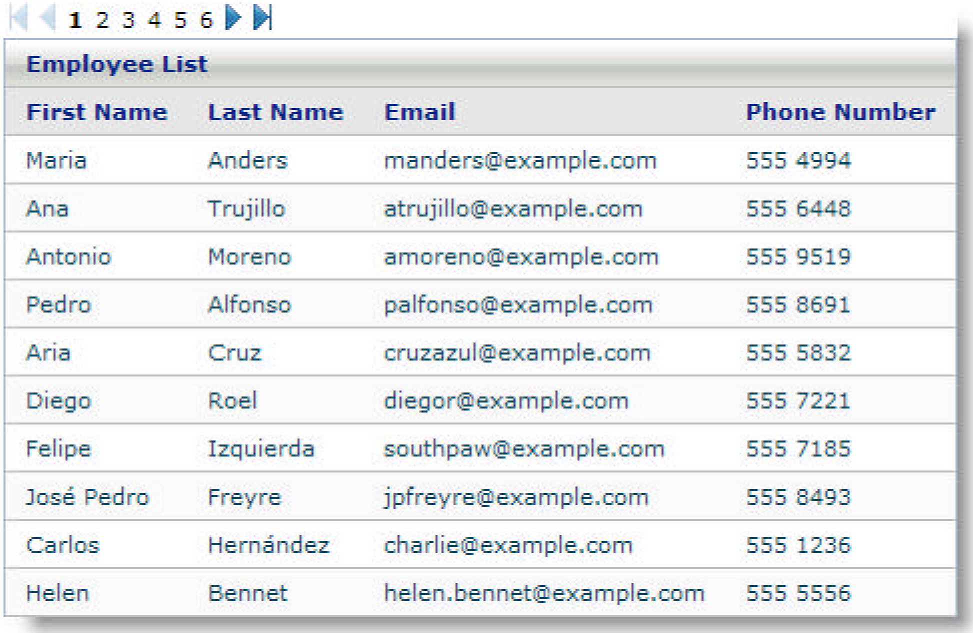
The following is a list of the *WebGrid*™ components:

- GridView
- Column
- ColumnSelectRow
- RowItem

WebGrid components supports editing and a host of other features and functionalities such as:

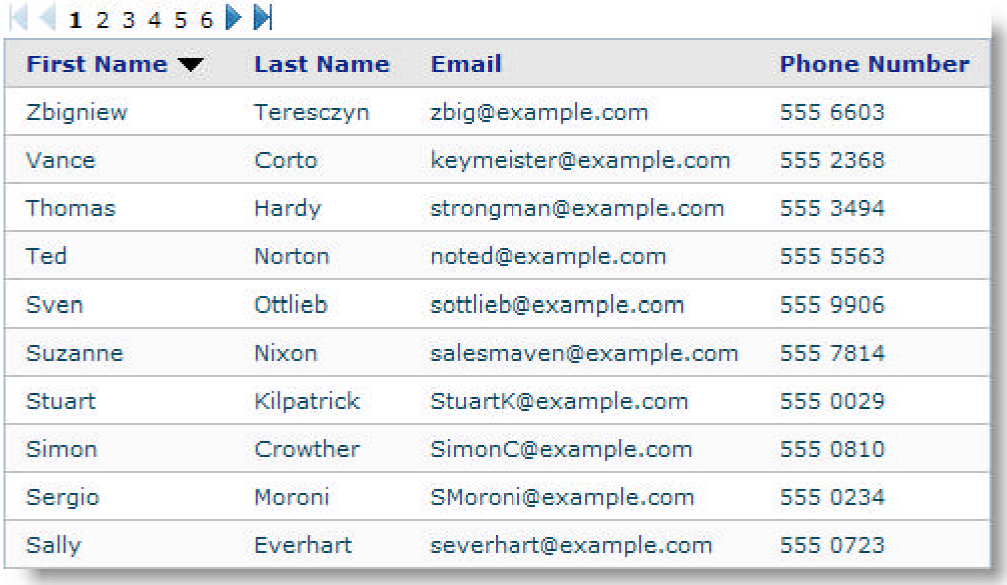
- Automatic Paging – Automatically display and handle the First, Previous, Next, and Last buttons. You only need to specify how many items you want to display. Smart Refresh™ transparently performs AJAX requests to reduce the amount of Web page refreshes.
- Automatic Sorting – You can set a per-column flag to automatically sort a specific column. Smart Refresh™ transparently performs AJAX requests to reduce the amount of Web page refreshes.
- High performance – Can handle large amounts of data being transferred to and from Web-based applications.
- Fully Customizable - WebGrid comes with a collection of professional, polished styles. You can also customize the look you want down to every detail because WebGrid is built on our AJAZ and UIElement bases framework. A UI element is any granular element of the WebGrid component, such as a row, column, hover, select column, etc. Every UIElement can be styled with your CSS styles or you can choose from the many styles that already come with the WebGrid components.
- Export to a CSV File - WebGrid allows you to export data to a Comma Separated Value (CSV) file format.
- Bound and Unbound mode – Supports binding to a JavaServer™ Faces DataMode, therefore it can be bound to Java™ arrays, Java lists, ResultSets, or JavaBeans™ by setting the WebGrid components for bound or unbound data rendering.

The first screen shot below shows the WebGrid components with automatic paging enabled. The second screen shot shows the WebGrid components with automatic sorting enabled.



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556



First Name ▼	Last Name	Email	Phone Number
Zbigniew	Teresczyn	zbig@example.com	555 6603
Vance	Corto	keymeister@example.com	555 2368
Thomas	Hardy	strongman@example.com	555 3494
Ted	Norton	noted@example.com	555 5563
Sven	Ottlieb	sottlieb@example.com	555 9906
Suzanne	Nixon	salesmaven@example.com	555 7814
Stuart	Kilpatrick	StuartK@example.com	555 0029
Simon	Crowther	SimonC@example.com	555 0810
Sergio	Moroni	SMoroni@example.com	555 0234
Sally	Everhart	severhart@example.com	555 0723

4.7.2 Getting Started with WebGrid

Click the links below for information on how to get up and running with the WebGrid components.

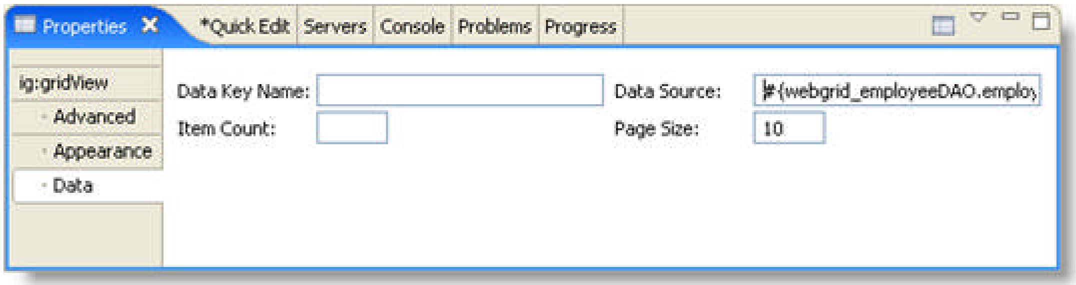
- **Creating WebGrid in Design View Using IBM RAD (Section 4.7.2.1)**
- **Creating WebGrid Using Code (Section 4.7.2.2)**

4.7.2.1 Creating WebGrid in Design View Using IBM RAD

Creating WebGrid using IBM® Rational® Application Developer for WebSphere 7.0 allows you to take full advantage of the JSP page designer tools palette. This helps you to create WebGrid components easier and quicker than ever before. You can drag the components from the palette to your JSP page, and control the layout and size of the components. Using the Properties window, you can edit the properties of your component and instantly see the changes in the design view, simplifying the editing process. Selecting a component from the design view populates the Properties window with the attributes and values of that component.

To create WebGrid in design view using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, drag the GridView component to your page.
2. In the Properties window, under ig:gridView, select Data.
3. Set the following properties:
   - Data Source: #{webgrid_employeeDAO.employees}
   - Page Size: 10



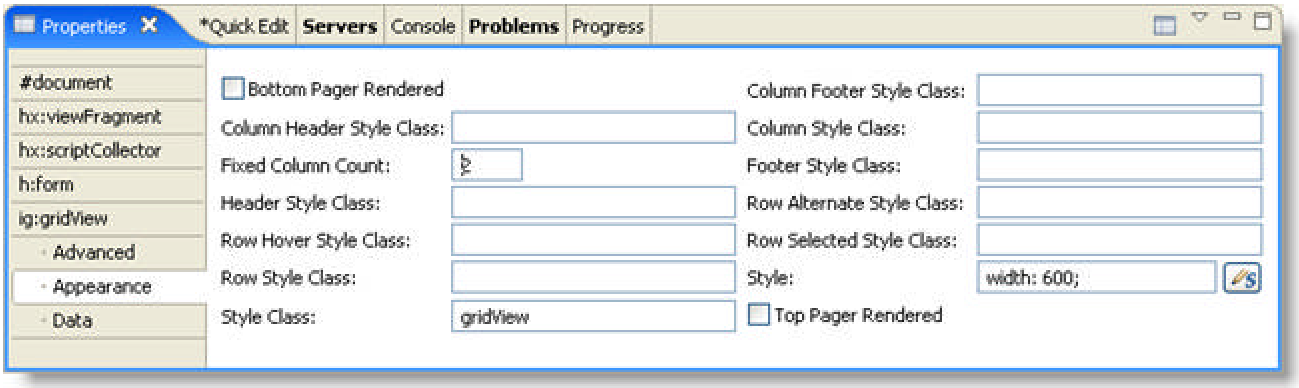
Data Key Name:

Item Count:

Data Source: #{webgrid_employeeDAO.employ

Page Size: 10

4. In the Properties window, under ig:gridView, click Appearance.
5. Set the following properties:
   - Fixed Column Count -- 2
   - Style -- Width: 600



□ Bottom Pager Rendered

Column Header Style Class:

Fixed Column Count: 2

Header Style Class:

Row Hover Style Class:

Row Style Class:

Style Class: gridView

Column Footer Style Class:

Column Style Class:

Footer Style Class:

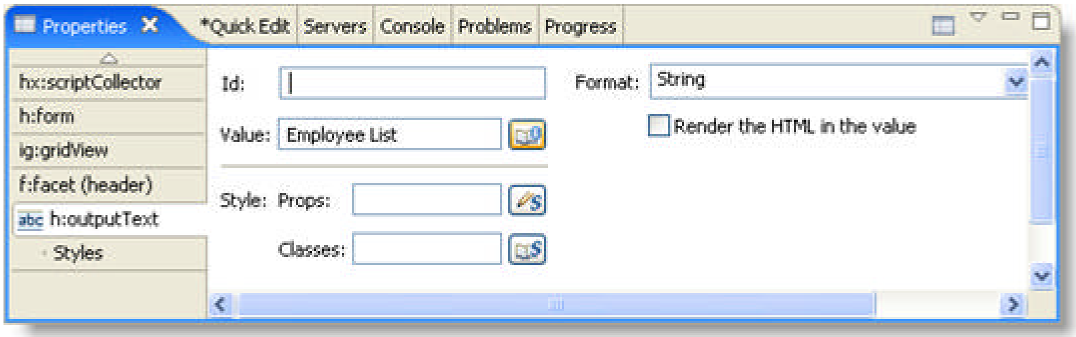
Row Alternate Style Class:

Row Selected Style Class:

Style: width: 600;

□ Top Pager Rendered

6. In the Design view, select the Grid Header and set the following property:
   - Value -- Employee List



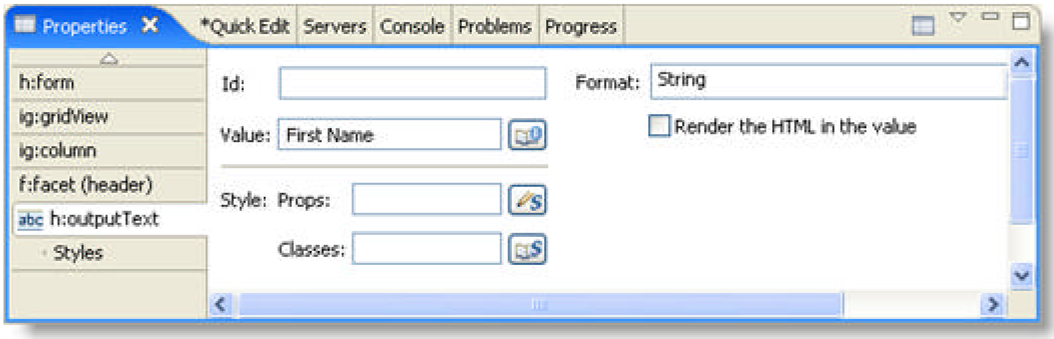
Id:

Value: Employee List

Format: String

□ Render the HTML in the value

7. The default GridView is created with two columns. From the palette, drag the Column component to your grid to add more columns.
8. In the Design view, for each column header, set the Value property:
   - For Column Header 1, set Value to First Name.
   - For Column Header 2, set Value to Last Name.
   - For Column Header 3, set Value to Email.
   - For Column Header 4, set Value to Phone Number.



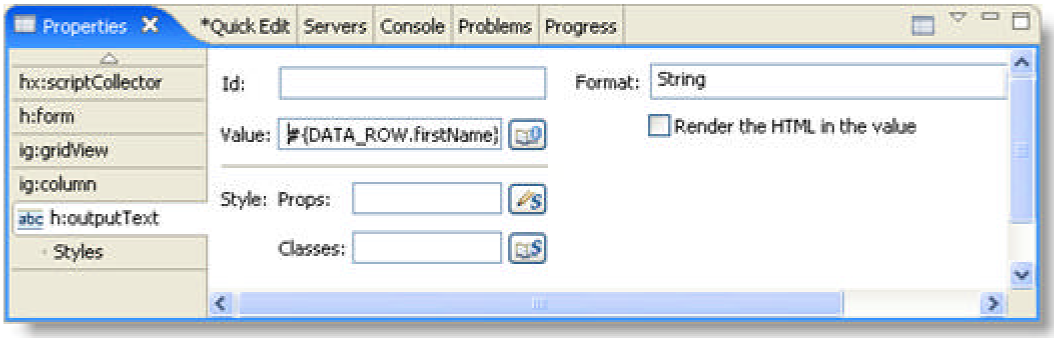
Id:

Value: First Name

Format: String

□ Render the HTML in the value

9. In the Design view, for each column, set the Value property:
   - For Column 1, set Value to #{DATA_ROW.firstName}.
   - For Column 2, set Value to #{DATA_ROW.lastName}.
   - For Column 3, set Value to #{DATA_ROW.email}.
   - For Column 4, set Value to #{DATA_ROW.phoneNumber}.



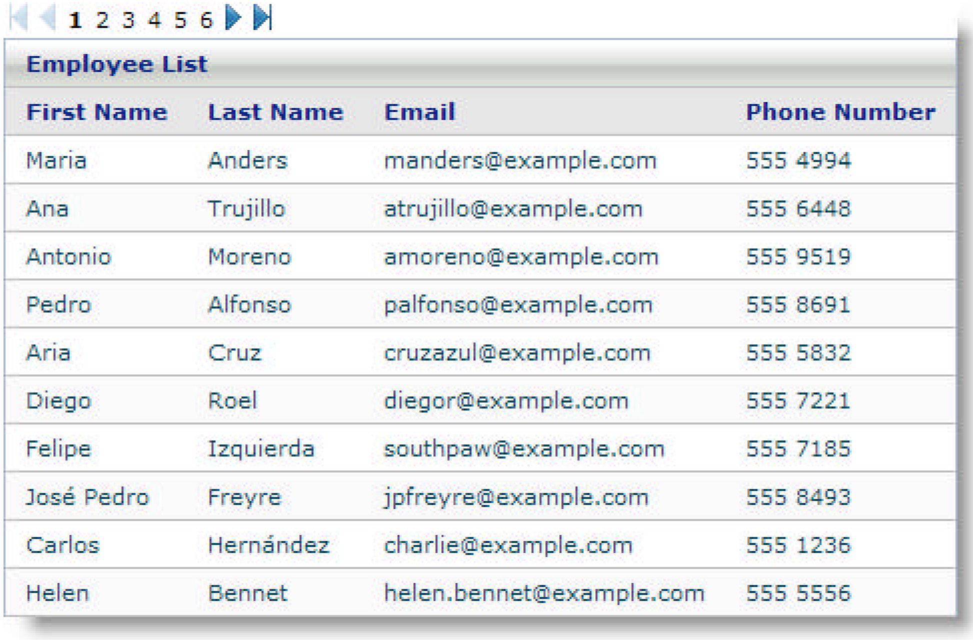
Id:

Value: #{DATA_ROW.firstName}

Format: String

□ Render the HTML in the value

10. Run the project. It should look similar to the grid below.



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

Related Topic

Creating WebGrid Using Code (Section 4.7.2.2)

4.7.2.2 Creating WebGrid Using Code

Creating WebGrid using code allows you to have full control over the design of your grid.

To create WebGrid using code:

1. Open the <ig:gridView> tag and set the following attributes, as shown in the JSP code below:
   - dataSource -- #{webgrid_employeeDAO.employees}
   - pageSize -- 10
   - fixedColumns -- 2
   - style -- width:600px

   In JSP:

```
<ig:gridView
        dataSource="#{webgrid_employeeDAO.employees}"
        pageSize="10"
        fixedColumns="2"
        style="width: 600px;">
```

2. Set the header of your grid to "Employee List", as shown in the JSP code below.
   In JSP:

```
<f:facet name="header">
    <h:outputText value="Employee List" />
</f:facet>
```

3. Set the column names of your grid to the following attributes, as shown in the JSP code below:
   - Column 1: First Name
   - Column 2: Last Name
   - Column 3: Email
   - Column 4: Phone Number

   In JSP:

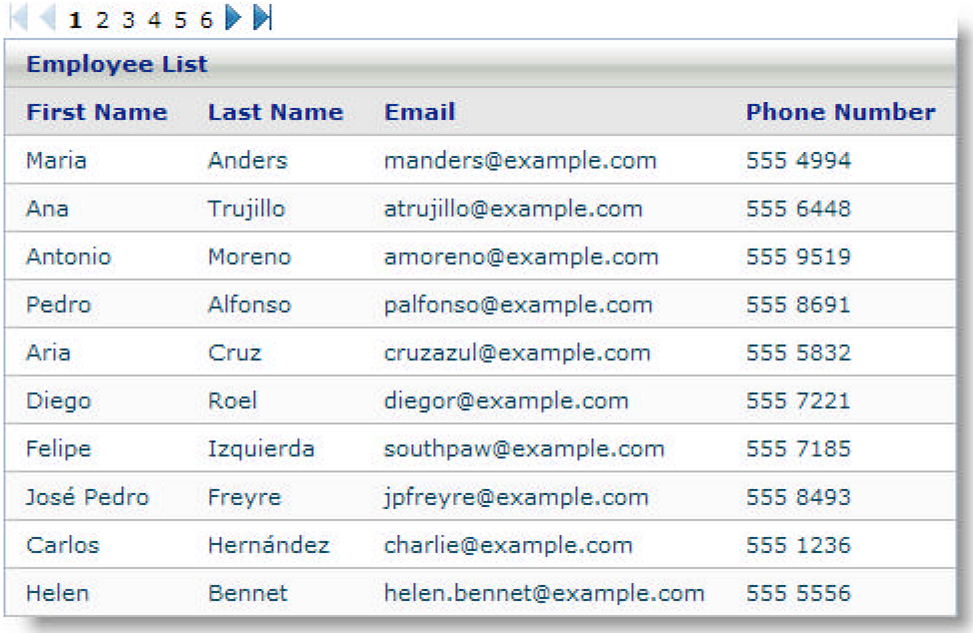
```
<ig:column>
    <f:facet name="header">
       <h:outputText value="First Name" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.firstName}" />
</ig:column>
<ig:column>
    <f:facet name="header">
       <h:outputText value="Last Name" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.lastName}" />
</ig:column>
<ig:column>
    <f:facet name="header">
       <h:outputText value="Email" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.email}" />
</ig:column>
```

```
<ig:column>
    <f:facet name="header">
       <h:outputText value="Phone Number" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.phoneNumber}" />
</ig:column>
```

4. Close the <ig:gridView> tag, as shown in the JSP code below.
   In JSP:

```
</ig:gridView>
```

5. Run the project. It should look similar to the grid below.



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

Related Topic

Creating WebGrid in Design View Using IBM RAD (Section 4.7.2.1)

4.7.3 Using WebGrid

This section is your gateway to important conceptual and task-based information that will help you to use the various features and functionalities provided by the *WebGrid* component.

- **Accessing Data (Section 4.7.3.1)**
- **Aggregate Functions (Section 4.7.3.2)**
- **Designing the Look and Feel (Section 4.7.3.3)**
- **Editing Data (Section 4.7.3.4)**
- **Navigating and Selecting (Section 4.7.3.5)**

4.7.3.1 Accessing Data

The main purpose of WebGrid is to input or display data. The first step is to bind the WebGrid component to a data source.

Click the links below for information on how to access data with the WebGrid components.

- **Data Binding Overview (Section 4.7.3.1.1)**
- **Bind a Hierarchical WebGrid to Data (Section 4.7.3.1.2)**
- **Bind WebGrid to Data (Section 4.7.3.1.3)**
- **Exporting Data to a CSV File (Section 4.7.3.1.4)**
- **Identify the Rows in a Collection (Section 4.7.3.1.5)**

4.7.3.1.1 Data Binding Overview

Data binding plays a pivotal role in WebGrid. The fundamental principle of WebGrid is to display, edit and manipulate data.

WebGrid can be bound to a data model through the dataSource attribute of the <ig:gridView> tag which points to a valid data model of a backing bean. The GridView component will cycle through each row in the list displaying each row of the grid.

The binding attribute of the <ig:gridView> tag is used when you want to use the grid as a property in the backing bean so that the grid can be manipulated in code.

Related Topic

Accessing Data (Section 4.7.3.1)

4.7.3.1.2 Bind a Hierarchical WebGrid to Data

WebGrid displays data in hierarchical views, allowing nested grids to be attached to a column. Expandable columns will have + or - icon allowing the end user to expand or collapse the columns. An expanded column will show a nested grid. Nested grids can also have expandable columns allowing for unlimited depth. Nested grids are separate object that can be independently sorted and paged as needed.

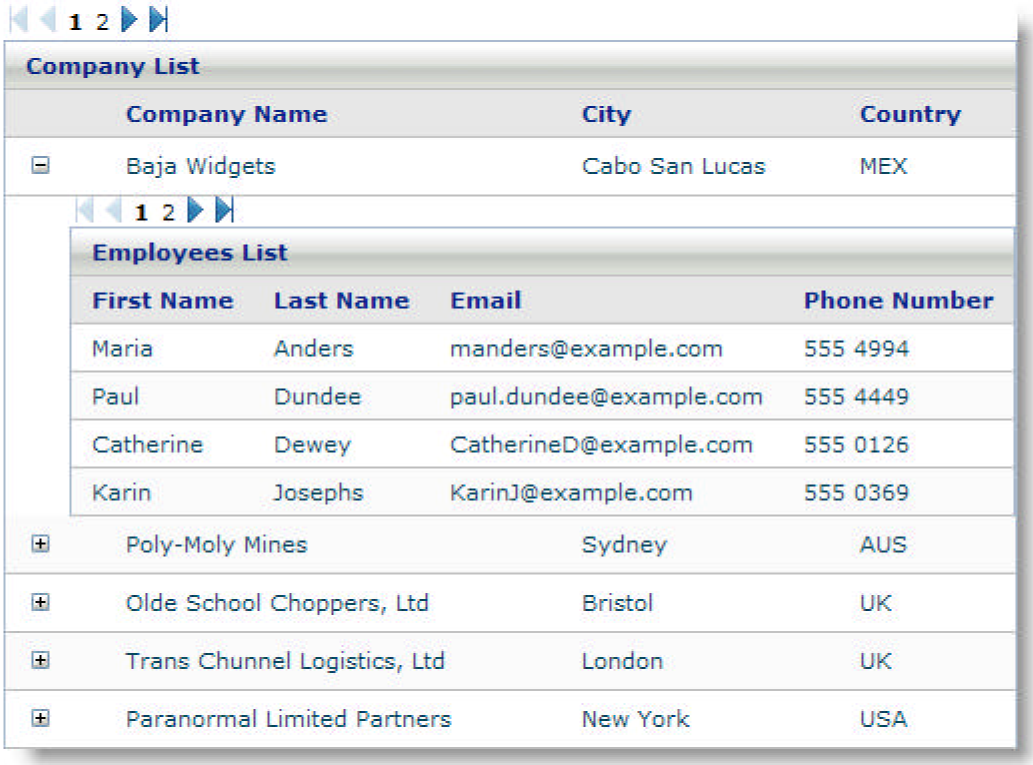
Create a facet called masterDetail and create the nested grid information within this tag.

To create a hierarchical grid.

1. Create a masterDetail facet within an <ig:column> tag.
   In JSP:

```
<ig:column>
  <f:facet name="header">
     <h:outputText value="Purchase Orders" />
  </f:facet>
<f:facet name="masterDetail">
     <ig:gridView dataSource="#DATA_ROW.orders" pagesize="10">
        <ig:column>
           <f:facet name="header">
              <h:outputText value="Order ID" />
           </f:facet>
           <h:outputText value="$DATA_ROW.orderId" />
        </ig:column>
     </ig:gridView>
</f:facet>
</ig:column>
```

The following screen shot shows a hierarchical grid.



Company List

	Company Name	City	Country
⊟	Baja Widgets	Cabo San Lucas	MEX

Employees List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Paul	Dundee	paul.dundee@example.com	555 4449
Catherine	Dewey	CatherineD@example.com	555 0126
Karin	Josephs	KarinJ@example.com	555 0369

	Company Name	City	Country
⊞	Poly-Moly Mines	Sydney	AUS
⊞	Olde School Choppers, Ltd	Bristol	UK
⊞	Trans Chunnel Logistics, Ltd	London	UK
⊞	Paranormal Limited Partners	New York	USA

4.7.3.1.3 Bind WebGrid to Data

Binding WebGrid to data populates your grid with the data from your data model.

To bind your grid to a data model.

1. Set the dataSource attribute of the <gridView> tag to point to your backing bean.
   In JSP:

```
<ig:gridView
    dataSource="#{webgrid_automaticSortingPage.employees}"
    pageSize="10"
    fixedColumnCount="2"
    style="width:600px">
```

2. Update the managed-beans.xml file to define the connection between the value of the dataSource attribute and your backing bean.
   In XML:

```
<managed-bean>
  <description>backing bean for the page automaticSorting</description>
  <managed-bean-name>webgrid_automaticSortingPage</managed-bean-name>
  <managed-bean-class>
        com.infragistics.faces.samples.webgrid.automaticSorting.Index
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

3. Set the value attribute of the <h:outputText> tag nested within the <ig:column> tag to the property value of the object model.
   In JSP:

```
<ig:column sortBy="firstName">
       <f:facet name="header">
           <h:outputText value="First Name" />
       </f:facet>
       <h:outputText value="#{DATA_ROW.firstName}" />
</ig:column>
```

4.7.3.1.4 Exporting Data to a CSV File

This feature allows data to be exported from a *WebGrid*™ to a Comma Separated Value (CSV) file. The data to be exported can be selected in either of the following ways.

- All the data on the current page of the WebGrid component can be exported.
- The end user can select specific rows to be exported.
- The entire WebGrid can be exported

> **Note** The only format currently supported is CSV.

As shown in the following code example, the Export method accepts different parameters depending on the data to be exported.
In Java:

```
grid.Export(RowToExport rowToExport, ExportFormat exportFormat)
```

- RowToExport is a constant representing the rows to be exported such CURRENT_ROWS, SELECTED_ROWS or ENTIRE_GRID.
- ExportFormat is a constant representing the format to export to.

To export WebGrid data to a CSV file:

The following example assumes that you have a grid created with Row Selection enabled.

For information on how to create a grid, see **Getting Started with WebGrid (Section 4.7.2)**. For information on how to enable row selection, see **Identify the Rows in a Collection (Section 4.7.3.1.5)**.

1. Add a command button to your page and add an action listener to it, as shown in the example JSP code below.
In JSP:

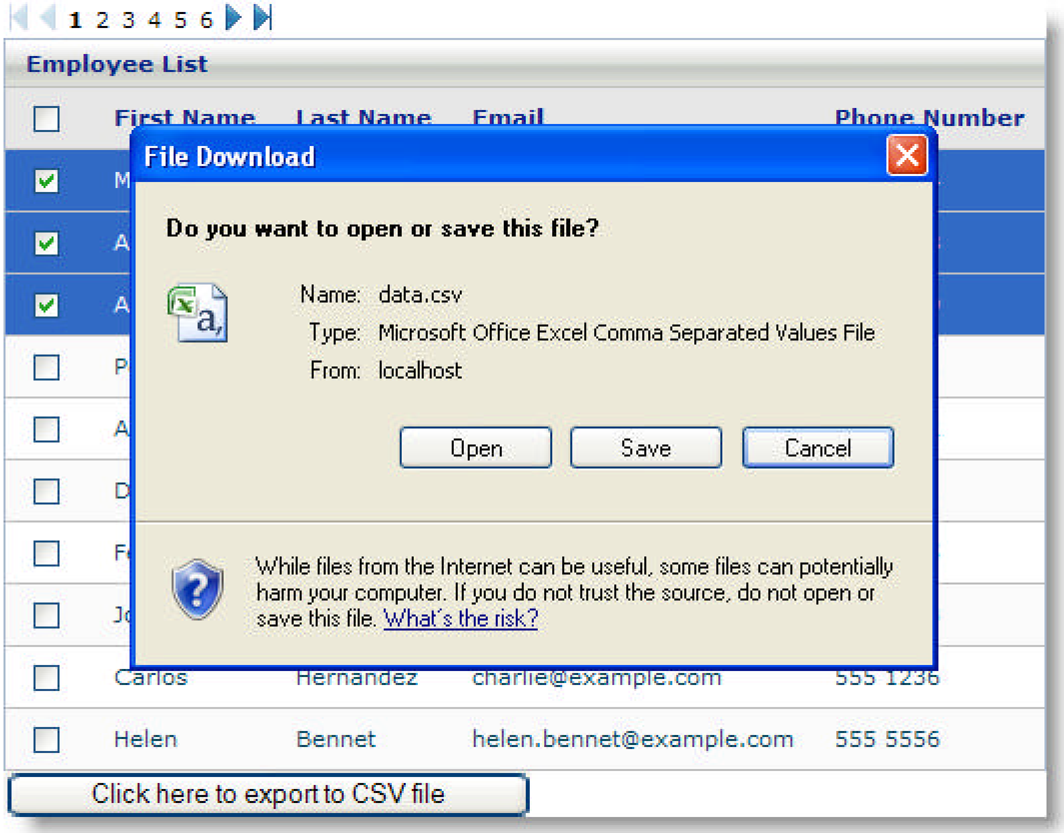
```
<h:commandButton
         value="Export"
         actionListener="#{webgrid_exportPage.onClick}">
</h:commandbutton>
```

2. Create the onClick method in your backing bean which calls the export method, as shown in the example JSP code below.
In Java:

```
public void onClick(ActionEvent event) {
// ask the grid to export its data in CSV format
 FacesContext context = FacesContext.getCurrentInstance();
// export selected rows if any
if (getGrid().getSelectedRows().size()>0) {
    getGrid().export(DataToExport.SELECTION, ExportFormat.CSV);
 }
else {
// otherwise export the current page
    getGrid().export(DataToExport.CURRENT, ExportFormat.CSV);
 }
 context.responseComplete();
```

```
    }
```

3. Save and run your project.
4. After you select specific rows in your grid and click Export, the data will be exported to a CSV file.



Employee List

□	First Name	Last Name	Email	Phone Number
☑	M			
☑	A			
☑	A			
□	P			
□	A			
□	D			
□	F			
□	Jo			
□	Carlos	Hernandez	charlie@example.com	555 1236
□	Helen	Bennet	helen.bennet@example.com	555 5556

File Download

Do you want to open or save this file?

Name: data.csv
Type: Microsoft Office Excel Comma Separated Values File
From: localhost

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

4.7.3.1.5 Identify the Rows in a Collection

An enumerator constant, DATA_ROW, has been defined for use with any NetAdvantage for JSF components that can be bound to a data collection. The DATA_ROW constant is used to represent the row number of the data in the collection.

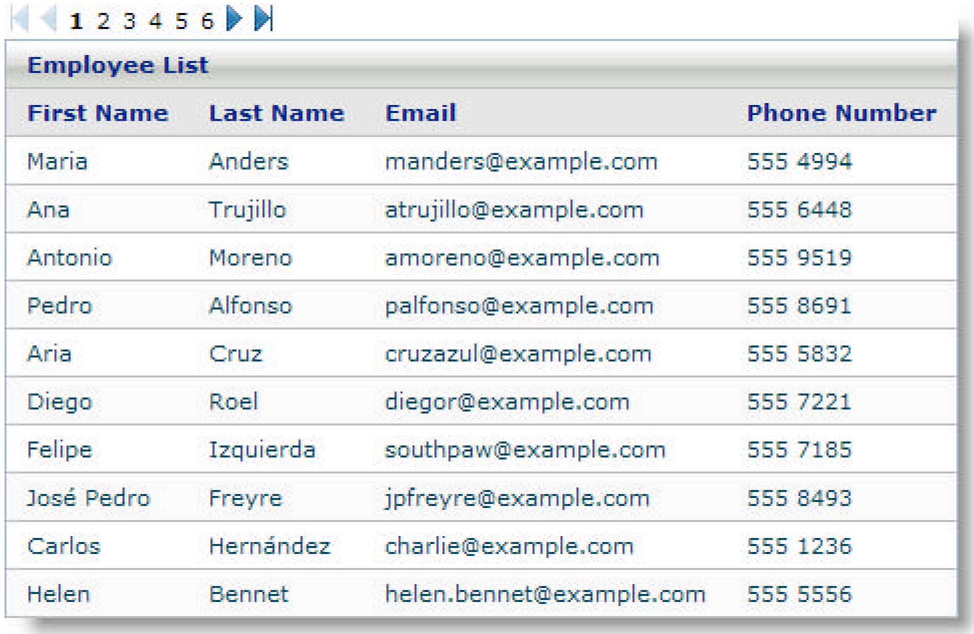
As an example, consider a WebGrid component that is bound to an ArrayList of employee data. The backing bean has a getEmployees() method that is used to get the ArrayList. Each element in the ArrayList is of type Employee which is a simple JavaBean with getters and setters for its fields, which include firstName, lastName, email, and phoneNumber.

The DATA_ROW constant can be used in valid value binding expression. See the **Binding JSF Components to Data (Section 4.2)** for more information about value binding expressions.

In JSP:

```
<ig:gridView
        dataSource="#{webgrid_employeeDAO.employees}"
        pagesize="10">
        <f:facet name="header">
                <h:outputText value="Employee List" />
        </f:facet>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="First Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.firstName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Last Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.lastName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Email" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.email}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Phone Number" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.phoneNumber}" />
        </ig:column>
</ig:gridView>
```

When the above code is executed with sample data, the following is the resulting WebGrid.



1 2 3 4 5 6

Employee List

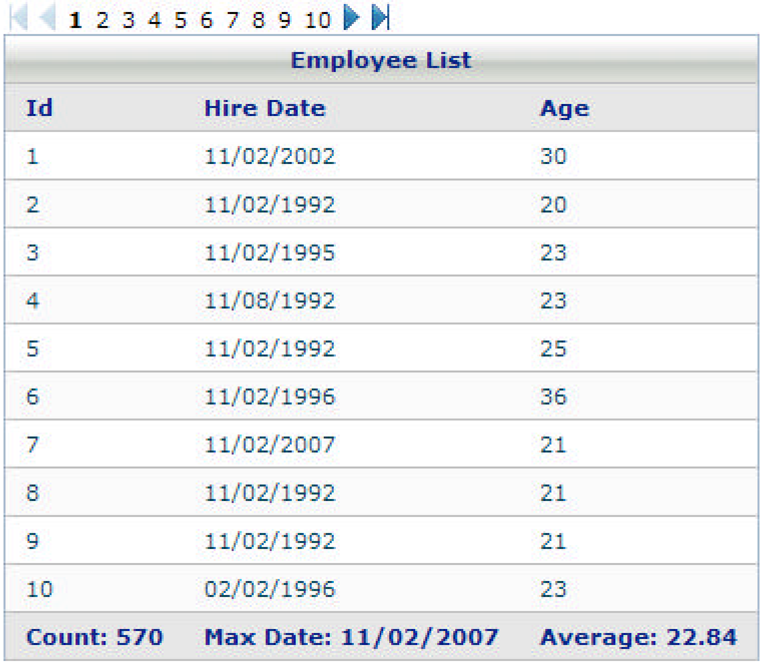
First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

4.7.3.2 Aggregate Functions

A key feature of the *WebGrid* component is that you can display values in the column footer that are calculated using aggregated functions. For example, you can display a running total of a column or the maximum value of a column in the footer.

This topic will demonstrate how to display calculations in the column's footer, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



1 2 3 4 5 6 7 8 9 10

Employee List		
Id	**Hire Date**	**Age**
1	11/02/2002	30
2	11/02/1992	20
3	11/02/1995	23
4	11/08/1992	23
5	11/02/1992	25
6	11/02/1996	36
7	11/02/2007	21
8	11/02/1992	21
9	11/02/1992	21
10	02/02/1996	23
Count: 570	**Max Date: 11/02/2007**	**Average: 22.84**

You can perform the following functions on the values of a column:

- SUM
- COUNT
- AVERAGE
- MAXIMUM
- MINIMUM

An aggregate function is applied to every row in the data set that the grid is bound to, regardless if those rows are displayed to the end user or not. For example, if you implement paging on your WebGrid to display ten rows per page to your end user, the value in the column footer is calculated using every row in the data set, not just the the rows that are displayed per page.

When using a hierarchical grid, the calculated values displayed in the column footer are associated with the parent level only.

When your end user exports WebGrid to a Comma Separated Value (CSV) file, the result displayed in the column footer will also be exported.

In order to take advantage of this feature, you simply add the <ig:gridAgFunction> tag to the footer facet of the column you want the calculations to be displayed. The applyOn attribute specifies which column you want the calculation performed on. You can specify the aggregate function by setting the type attribute of <ig:gridAgFunction>.

As with all of our components the aggregate function component can be easily styled to suit your needs. For example, you may want the results to be displayed in a different font. You can style the appearance of the results using the style and styleClass attributes of the <ig:gridAgFunction>

The following code demonstrates how to calculate the average value of a column.

To enable grid activation using code:

In JSP:

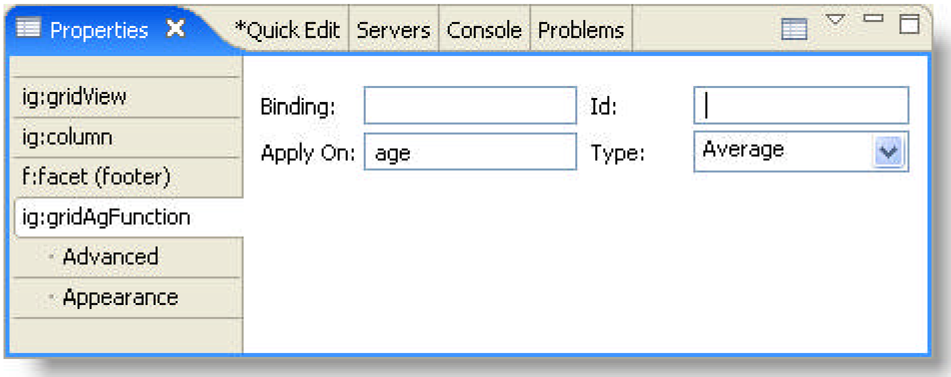
```
<ig:column>
   <f:facet name="header">
      <h:outputText value="Age" />
   </f:facet>
   <h:outputText value="#{DATA_ROW.age}" />
   <f:facet name="footer">
      <ig:gridAgFunction id="af01" applyOn="age" type="average" styleClass="igGrid_A
   </f:facet>
</ig:column>
```

In Java:

```
 HtmlGridAgFunction agFunction = new HtmlGridAgFunction();
 agFunction.setApplyOn("age");
 agFunction.setType("average");
 agFunction.setStyleClass("igGrid_AgFunction");
 column.setFooter(agFunction);
```

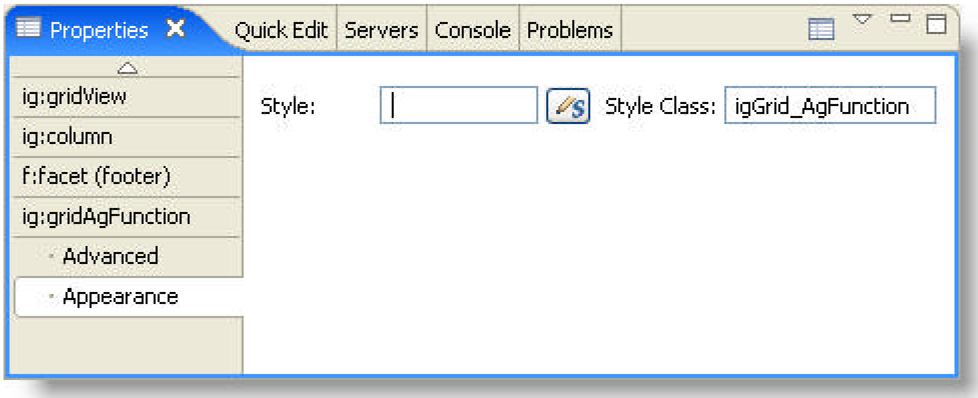
To enable grid activation using RAD:

1. From the palette, drag the GridAgFunction component to the column on your WebGrid component where you want the calculation to be displayed.
2. In the Properties window, under ig:gridAgFunction, set the following properties:
   - Apply On -- Set this property to the column you want the calculation to be performed on.
   - Type -- Set this property to the type of calculation you want performed.



Binding:
Id:
Apply On: age
Type: Average

3. In the Properties window, under ig:gridAgFunction, click Appearance.
4. Set the Style Class property to your style class



Style:

Style Class: igGrid_AgFunction

4.7.3.3 Designing the Look and Feel

The look and feel of a component is very important when creating an application. The "look" implies how a component visually appears, but the "feel" implies how a component acts; thus, you receive an appealing visual style coupled with specific functionality.

You can easily modify the visual appearance and behavior of WebGrid by changing the grid's color scheme, applying vertical and horizontal scroll bars to your grid, and enabling resizable columns.

Click the following links to view topics written specifically to help you design the look and feel of the WebGrid components.

- **Column Moving (Section 4.7.3.3.1)**
- **Fixed Columns (Section 4.7.3.3.2)**
- **Resizable Columns (Section 4.7.3.3.3)**
- **Setting the Width of a Column (Section 4.7.3.3.4)**
- **Horizontal Scroll Bars (Section 4.7.3.3.5)**
- **Vertical Scroll Bars (Section 4.7.3.3.6)**
- **Force Vertical Scroll Bars (Section 4.7.3.3.7)**

Click the following link for information on the designing the look and feel.

- **Designing the Look and Feel (Section 5)**

4.7.3.3.1 Column Moving

The *WebGrid*™ component has built-in functionality for allowing the user to move columns. Your end user can move a column at run-time by selecting the column and dragging it into it's new position.

By default, column moving is enabled. There may be certain situations where you do not want your end user to move the columns within your WebGrid. For example if you have a WebGrid showing project details, you may not want to allow your end user to move the End Date column before the Start Date column.

You can set column moving to be disabled on the entire WebGrid or on a per column basis.

To disable column moving on the entire WebGrid, you simply set the movableColumns attribute of the <ig:gridView> to False.

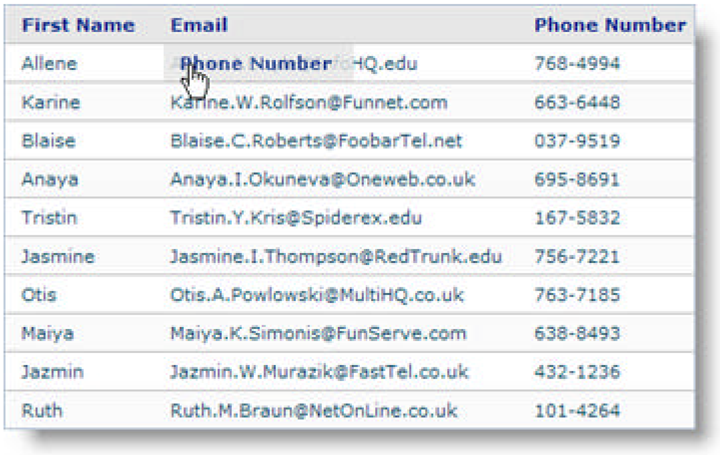
If you want to disable a certain column from moving you just set the movable attribute of the <ig:column> or <ig:columnSelectRow> to False.

The attribute movableColumns of <ig:gridView> has a higher priority than the attribute movable of <ig:column>. For example if you set the attribute movableColumns of <ig:gridView> to False, then you can not move the columns even if you set the movable attribute of <ig:column> to True. However, if you set the attribute movableColumns of <ig:gridView> to True, you can disable moving individual columns by setting the movable attribute of <ig:column> to False.

> **Note:** If you disable column moving on your WebGrid component, then the pin icon that allows your end user to set a column as fixed is hidden. Also your end user will not be allowed to move columns between fixed and unfixed columns.

This topic will demonstrate how to disable column moving on your WebGrid, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name	Email	Phone Number
Allene	HQ.edu	768-4994
Karine	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Jasmine.I.Thompson@RedTrunk.edu	756-7221
Otis	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
Maiya	Maiya.K.Simonis@FunServe.com	638-8493
Jazmin	Jazmin.W.Murazik@FastTel.co.uk	432-1236
Ruth	Ruth.M.Braun@NetOnLine.co.uk	101-4264

To disable column moving using code:

<u>In the entire WebGrid component:</u>

In JSP:

```
<ig:gridView
    dataSource="#{webgrid_movableColumnsPage.employees}"
    pageSize="10"
    movableColumns="false" />
```

In Java:

```
  HtmlGridView gridView = new HtmlGridView();
  gridView.setMovableColumns(false);
```

<u>Per column basis:</u>

In JSP:

```
<ig:gridView
    dataSource="#{webgrid_movableColumnsPage.employees}"
    pageSize="10"
    movableColumns="true" />

    <ig:column movable="false">
        <f:facet name="header">
            <h:outputText value="First Name" />
        </f:facet>
        <h:outputText value="#{DATA_ROW.firstName}" />
    </ig:column>
```

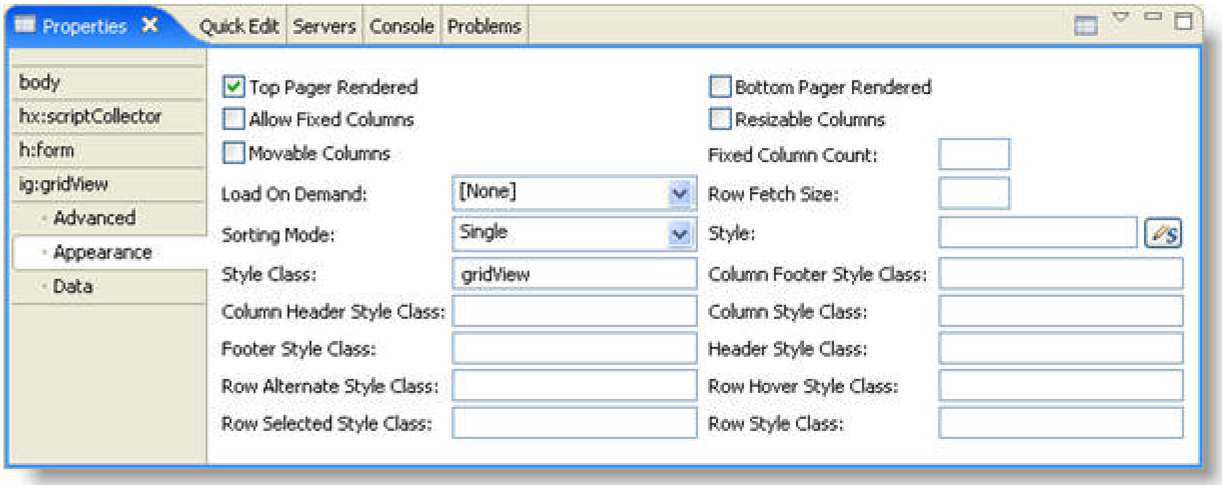
In Java:

```
  HtmlGridView gridView = new HtmlGridView();
  gridView.setMovableColumns(true);

  Column columnMove = new Column();
  columnMove.setMovable(false);
```

To disable column moving using RAD:

<u>In the entire WebGrid component:</u>

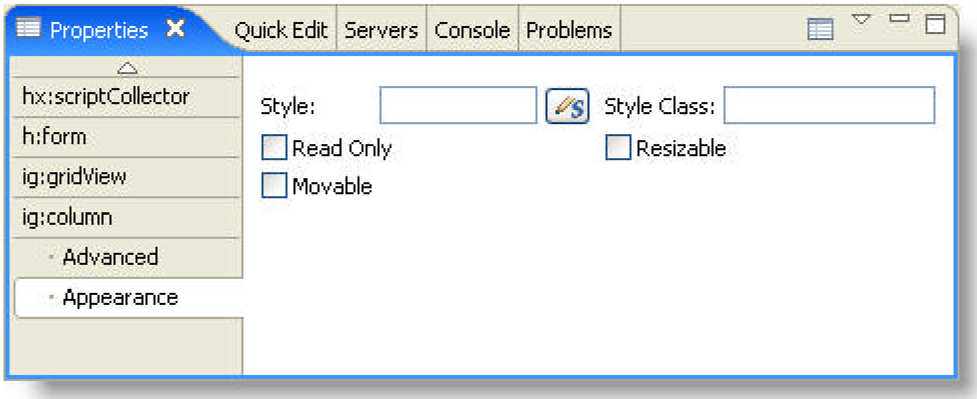
1. In Design view, select the WebGrid component.
2. In the Properties window, under ig:gridView, select Appearance.
3. Deselect the Movable Columns check box.



☑ Top Pager Rendered
□ Bottom Pager Rendered
□ Allow Fixed Columns
□ Resizable Columns
□ Movable Columns
Fixed Column Count:
Load On Demand: [None]
Row Fetch Size:
Sorting Mode: Single
Style:
Style Class: gridView
Column Footer Style Class:
Column Header Style Class:
Column Style Class:
Footer Style Class:
Header Style Class:
Row Alternate Style Class:
Row Hover Style Class:
Row Selected Style Class:
Row Style Class:

<u>Per column basis:</u>

1. In Design view, select the Column component.
2. In the Properties window, under ig:column, select Appearance.
3. Deselect the Movable check box.



Style:
Style Class:
□ Read Only
□ Resizable
□ Movable

4.7.3.3.2 Fixed Columns

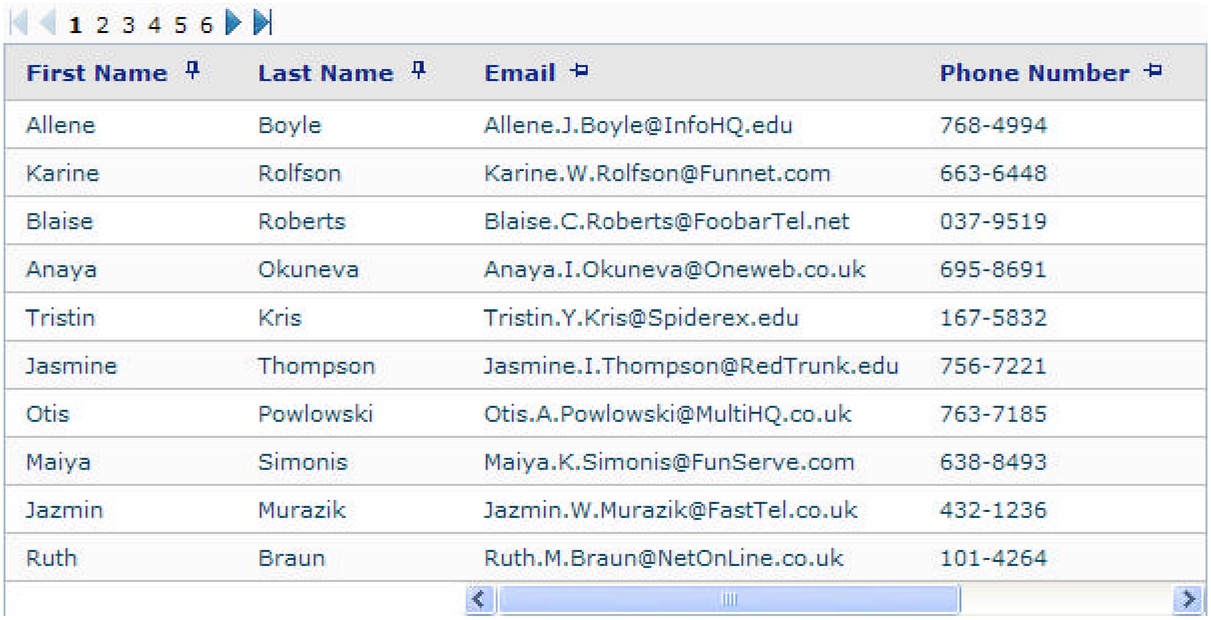
One of the key features of WebGrid™ is the fixed column functionality. This feature allows you to create a fixed (non-scrolling) column so that it remains in view when the grid is scrolled horizontally.

When the end user clicks the column indicator in the header, the column will shift to the left of the grid and will remain fixed during horizontal scrolling.

By default, the fixed column functionality is turned off.

This topic will demonstrate how to enable fixed columns on your WebGrid, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221
Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493
Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236
Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264

To enable fixed columns using code:

1. Set the following attributes of the <ig:gridView> tag, as shown in the JSP code below
   - allowFixedColumns -- true
   - fixedColumnCount -- 2

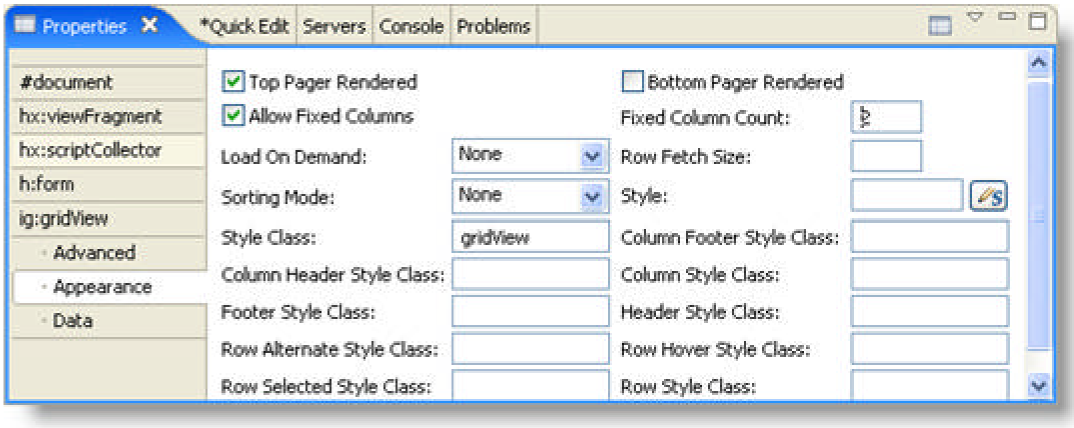
   In JSP:

```
<ig:gridView
     dataSource="#{webgrid_fixedColumnsPage.employees}"
     pageSize="10"
     allowFixedColumns="true"
     fixedColumnCount="2"
     style="height: 200px; width: 600px;" />
```

To enable fixed columns using RAD:

1. In Design view, select the WebGrid component.
2. In the Properties window, under ig:gridView, select Appearance.

3. Set the following properties:
   - Allow Fixed Columns -- true
   - Fixed Column Count -- 2



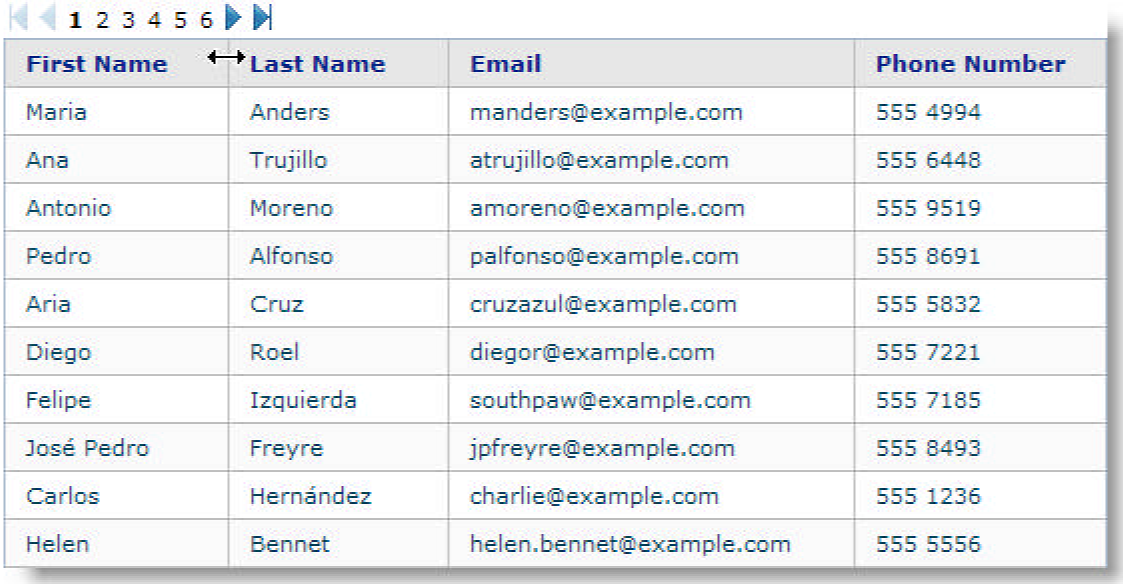
☑ Top Pager Rendered	□ Bottom Pager Rendered
☑ Allow Fixed Columns	Fixed Column Count: 2
Load On Demand: None	Row Fetch Size:
Sorting Mode: None	Style:
Style Class: gridView	Column Footer Style Class:
Column Header Style Class:	Column Style Class:
Footer Style Class:	Header Style Class:
Row Alternate Style Class:	Row Hover Style Class:
Row Selected Style Class:	Row Style Class:

4.7.3.3.3 Resizable Columns

One of the key features of the *WebGrid* components is the ability to resize the grid. With this feature, the end user can expand or shrink the columns of the grid for better usability. For example, if the data in a cell is too long for the user to read, they can expand the column to see all the information in the cell.

When you enable the resizing of columns in a grid, the end user can move the mouse pointer over the column separator in the header to drag the separator and resize the column. Each column is resized independently, and does not affect the other columns.

This topic will demonstrate how to enable column resizing, as illustrated in the screen shot below.



First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

To enable column resize:

1. You can configure the resizing of columns globally for the entire grid.
   Set the resizableColumns attribute of the <ig:gridView> tag to true:

   In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
     fixedColumnCount="2"
     resizableColumns="true"
     style= "width: 600px">
```

2. You can configure each column to be resizable.
   Set the resizable attribute of the <ig:column> tag to true:

   In JSP:

```
<ig:column resizable="true">
    <f:facet name="header">
       <h:outputText value="First Name" />
    </f:facet>
       <h:outputText value="#{DATA_ROW.firstName}" />
 </ig:column>
```

NetAdvantage for JSF

4.7.3.3.4 Setting the Width of a Column

There may be certain cases where the content in your column is extremely long and you want to lessen the width of the column containing the data.

To achieve this you have to explicitly tell the browser what to do. This can be done using the overflow style.

The following code demonstrates how to set the width of a column to 100px.

In JSP:

```
<ig:column resizable="false">
   <f:facet name="header">
      <h:outputText value="e-mail"></h:outputText>
   </f:facet>
   <h:outputText value="#{DATA_ROW.email}"
                 style="text-overflow:ellipsis;
                 overflow:hidden;
                 white-space:nowrap;
                 width:100px;
                 display:block;">
   </h:outputText>
</ig:column>
```

4.7.3.3.5 Horizontal Scroll Bars

When the content of *WebGrid*™ exceeds the specified width, a horizontal scroll bar is displayed.

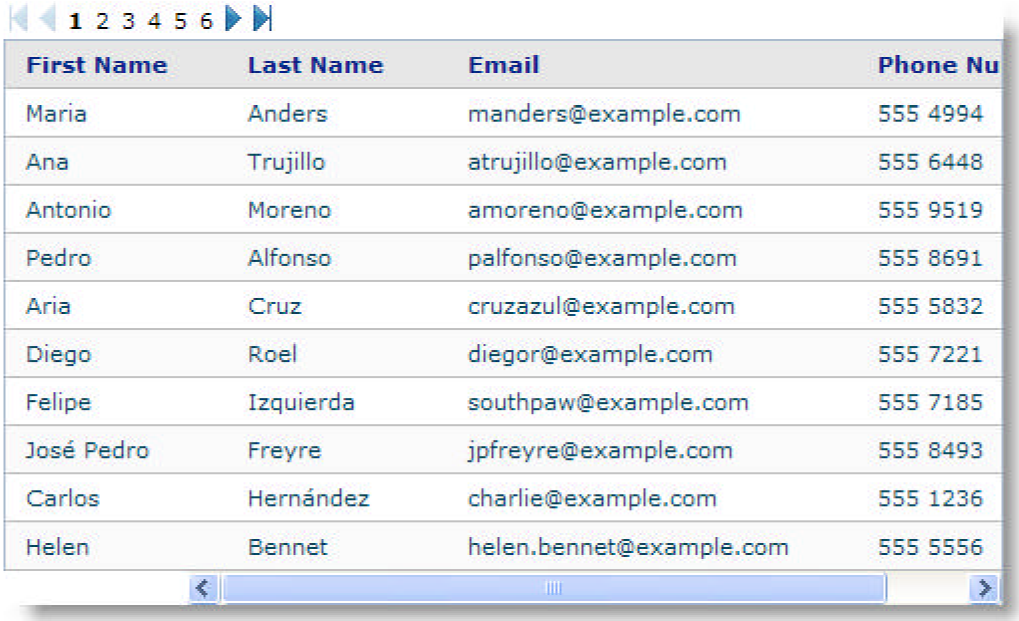
The horizontal scroll bar also appears when resizable columns is enabled on the grid or the columns.

If you are creating a grid with a large number of columns, the horizontal scroll bar will allow your end users to easily scroll through the columns of the grid.

You can specify certain columns to be fixed, meaning that these columns are not scrollable. For example, if you create a grid with a large number of columns and enable horizontal scrolling, and the end user is scrolls across the grid, the data may not hold much meaning unless the end user can see the data in the first column. For this reason, you can set the first column to be fixed.

This topic will demonstrate how to enable horizontal scroll bars, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



1 2 3 4 5 6

First Name	Last Name	Email	Phone Nu
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

To enable horizontal scroll bars using code:

1. Set the width attribute of the <ig:gridView> tag, as shown in the example code below.

   In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
     resizableColumns="true"
     style= "width: 600px; height: 200px">
```

2. If you want to a specific column to be fixed when the end user scrolls the grid, set the fixedColumnCount attribute:

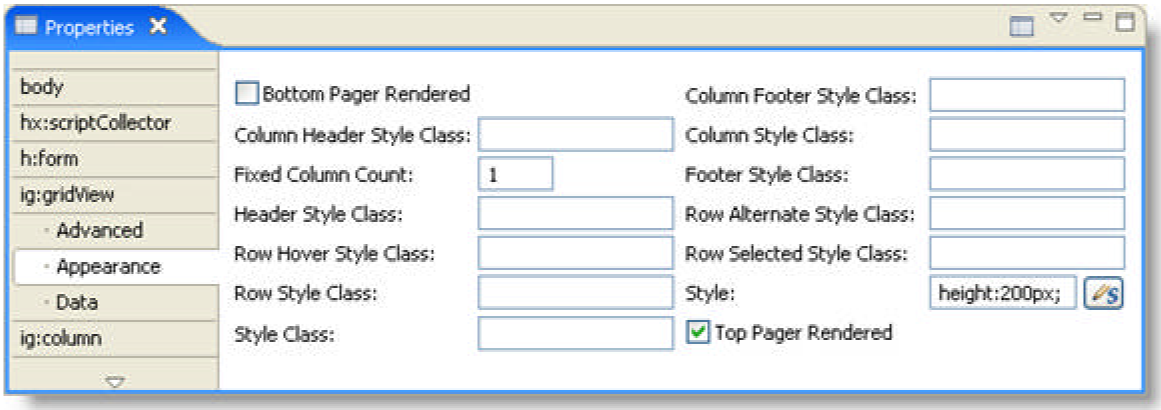
   In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
```

```
        fixedColumnCount="2"
        resizableColumns="true"
        style= "width: 600px; height: 200px">
```

To enable horizontal scroll bars with fixed columns at design time using RAD:

1. In Design view, select the grid component on your page.
2. In the Properties window, select the Appearance tab under the ig:gridView header.
3. Set the following properties:
   - Style: width:600px
   - Fixed Column Count: 1



□ Bottom Pager Rendered

Column Header Style Class:

Fixed Column Count: 1

Header Style Class:

Row Hover Style Class:

Row Style Class:

Style Class:

Column Footer Style Class:

Column Style Class:

Footer Style Class:

Row Alternate Style Class:

Row Selected Style Class:

Style: height:200px;

☑ Top Pager Rendered

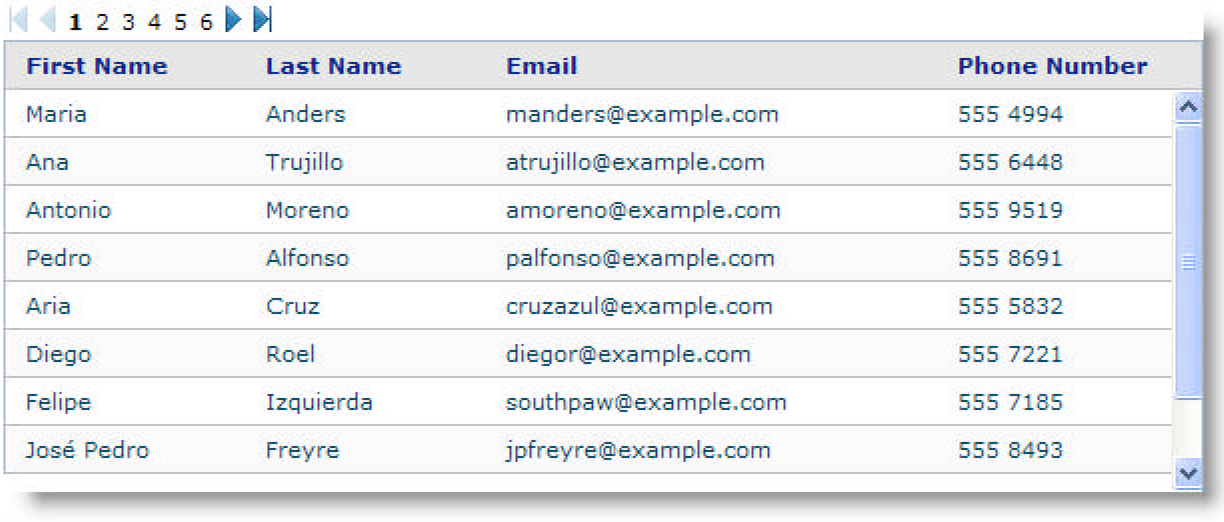
4.7.3.3.6 Vertical Scroll Bars

Starting in the 2007 Volume 2 release, when the content of *WebGrid*™ exceeds the specified height, a vertical scroll bar is displayed. If the height attribute of the <ig:gridView> is not set, the vertical scroll bar will not appear. When scrolling through a grid, the Header and Footer of the grid are stationary.

If you are creating a large grid, the vertical scroll bar will allow your end users to easily scroll through the data of the grid.

This topic will demonstrate how to enable vertical scroll bars, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493

To enable vertical scroll bars using code:

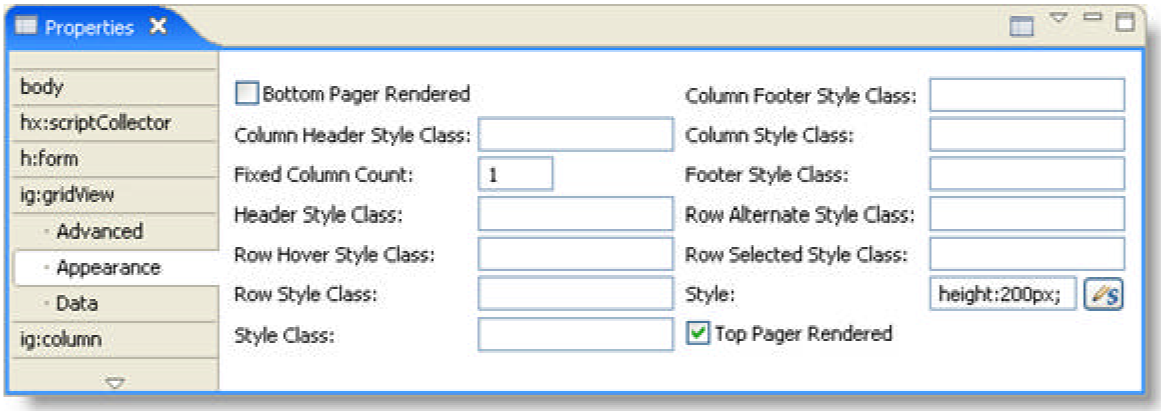
1. Set the height attribute of the <ig:gridView> tag, as shown in the example code below:

   In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
     fixedColumnCount="2"
     resizableColumns="true"
     style= "width: 600px; height: 200px">
```

To enable vertical scroll bars at design time using RAD:

1. From the palette, drag a GridView component on to your page.
2. In Design view, select the grid component on your page.
3. In the Properties window, select the Appearance tab under the ig:gridView header.
4. Set the height to 200px.



□ Bottom Pager Rendered

Column Header Style Class:

Fixed Column Count: 1

Header Style Class:

Row Hover Style Class:

Row Style Class:

Style Class:

Column Footer Style Class:

Column Style Class:

Footer Style Class:

Row Alternate Style Class:

Row Selected Style Class:

Style: height:200px;

☑ Top Pager Rendered

4.7.3.3.7 Force Vertical Scroll Bars

You can force vertical scroll bars to appear on your grid regardless of the value of the height attribute. To force vertical scroll bars you simply set the forceVerticalScrollBar attribute of the <ig:gridView> tag to True. The default value for this attribute is False.

The following code example demonstrates how to force vertical scroll bars on your grid.

In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
     resizableColumns="true"
     forceVerticalScrollBar ="true">
```

4.7.3.4 Editing Data

Click the following links to view topics written specifically to help you edit data using the WebGrid component.

- **Cell Editing (Section 4.7.3.4.1)**
- **Cell Editing Properties (Section 4.7.3.4.2)**

4.7.3.4.1 Cell Editing

Occasionally, you might use a database purely for informational purposes, never modifying the data. However, when this is not the case, you need a dependable and easy way for the end user to modify data. Modifying data is a task that the WebGrid™ component performs.

This topic explains how to allow your end users modify the data in a cell of WebGrid using valid JSF components such as Infragistics WebInput components.

To enable cell editing within your WebGrid, you simply add the <gridEditing> tag.

You can also set a column in an editable grid to be read only by setting the readOnly attribute of the <ig:column> tag.

When make your grid editable, a default editor (text box) is provided for each column. However, you can also specify your own editor. An editor is declared by setting the name attribute of the <f:facet> tag. Within the <f:facet> tag you can declare any editor, such as <ig:dateChooser>. The following code demonstrates how to do this.

In JSP:

```
<f:facet name="editor">
   <ig:dateChooser value="#{DATA_ROW.hireDate}"
       editMasks="MM/dd/yyyy"
       styleClass="igGridCellEditDate"/>
</f:facet>
```

> **Note:** When using a date chooser editor, the column type should be in the same date format as the date chooser editor. If the formats are different, use the <f:convertDataTime> tag to avoid errors when you are saving the changes.

To enable cell editing:

1. Open the <ig:gridView> tag and set the following attributes, as shown in the JSP code below:
   - dataSource -- #{webgrid_cellEditingPage.employees}
   - pageSize -- 10

   In JSP:

   ```
   <ig:gridView
           dataSource="#{webgrid_cellEditingPage.employees}"
           pagesize="10">
   ```

2. Declare tags to enable cell editing within the tags of gridView.
   In JSP:

   ```
   <ig:gridEditing enableOnMouseClick="single"
       cellValueChangeListener="#{webgrid_cellEditingPage.cellValueChangeListener}"
   ```

3. Set the header of your grid to "Employee List", as shown in the JSP code below.
   In JSP:

   ```
   <f:facet name="header">
       <h:outputText value="Employee List" />
   </f:facet>
   ```

4. Set the column names of your grid and the value of the columns to the following attributes, as shown in the JSP code below:
    - Column 1:
        - Column Name -- First Name
        - Column Value -- #{DATA_ROW.firstName}
    - Column 2:
        - Column Name -- Last Name
        - Column Value -- #{DATA_ROW.lastName}

    In JSP:

```
<ig:column>
    <f:facet name="header">
       <h:outputText value="First Name" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.firstName}" />
</ig:column>
<ig:column>
    <f:facet name="header">
       <h:outputText value="Last Name" />
    </f:facet>
    <h:outputText value="#{DATA_ROW.lastName}" />
</ig:column>
```

5. Set the column names of your grid and the value of the columns to the following attributes. Also set these columns to allow your end user to edit them using the EmailInput and DateChooser components, as shown in the JSP code below:
    - Column 1:
        - Column Name -- First Name
        - Column Value -- #{DATA_ROW.firstName}
        - Column Editor -- InputEmail
    - Column 2:
        - Column Name -- Last Name
        - Column Value -- #{DATA_ROW.lastName}
        - Column Editor -- DateChooser

    In JSP:

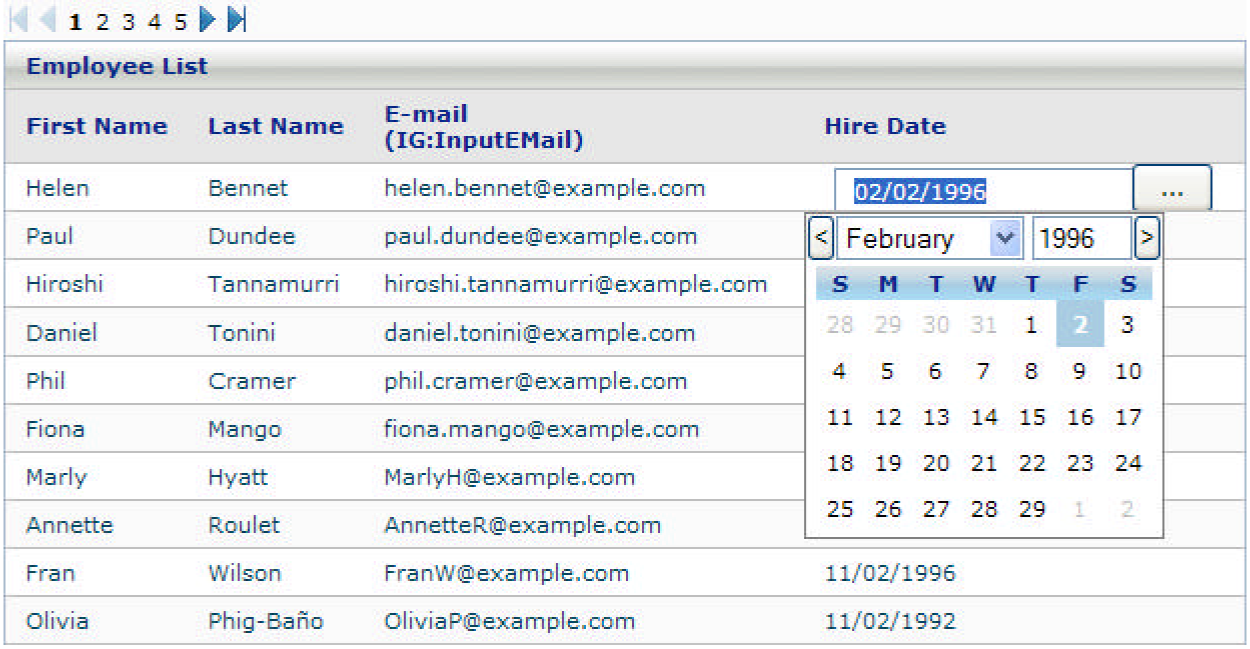
```
<ig:column style="width:200px">
   <f:facet name="header">
      <h:outputLabel value="E-mail(IG:InputEMail)" />
   </f:facet>
   <f:facet name="editor">
      <ig:inputEmail value="#{DATA_ROW.email}"
                styleClass="igGridCellEdit"
                style="width:180px;height:11px" />
   </f:facet>
   <h:outputText value="#{DATA_ROW.email}" />
</ig:column>
<ig:column style="width:200px">
```

```
   <f:facet name="header">
      <h:outputText value="Hire Date" />
   </f:facet>
   <f:facet name="editor">
      <ig:dateChooser value="#{DATA_ROW.hireDate}"
          editMasks="MM/dd/yyyy"
          styleClass="igGridCellEditDate" />
   </f:facet>
   <h:outputText value="#{DATA_ROW.hireDate}">
      <f:convertDateTime pattern="MM/dd/yyyy" />
   </h:outputText>
</ig:column>
```

6. Close the <ig:gridView> tag, as shown in the JSP code below.
   In JSP:

```
</ig:gridView>
```

7. Run the project. It should look similar to the grid below.



Employee List

First Name	Last Name	E-mail (IG:InputEMail)	Hire Date
Helen	Bennet	helen.bennet@example.com	02/02/1996
Paul	Dundee	paul.dundee@example.com	
Hiroshi	Tannamurri	hiroshi.tannamurri@example.com	
Daniel	Tonini	daniel.tonini@example.com	
Phil	Cramer	phil.cramer@example.com	
Fiona	Mango	fiona.mango@example.com	
Marly	Hyatt	MarlyH@example.com	
Annette	Roulet	AnnetteR@example.com	
Fran	Wilson	FranW@example.com	11/02/1996
Olivia	Phig-Baño	OliviaP@example.com	11/02/1992

< February 1996 >

S	M	T	W	T	F	S
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	1	2

Related Topic

Cell Editing Properties (Section 4.7.3.4.2)

4.7.3.4.2 Cell Editing Properties

This topic describes all the attributes within the cell editing tags.

Setting the <ig:gridEditing> tag allows your end users to edit your WebGrid component. This tag is nested within the <ig:gridView> tag.

The following is a list of all the attributes within the <ig:gridEditing> tag:

- **editMode** - Specifies the edit mode of the WebGrid component. You can set the editMode so that either a single cell is edited or a row is edited.
- **enableOnActive** - Grid activation must be enabled on your WebGrid for this attribute to work correctly. For more information, see **Grid Activation (Section 4.7.3.5.2)**. This attribute specifies if the cell will become enabled if your end user activates the cell.
- **enableOnKeyPress** - Grid activation must be enabled on your WebGrid for this attribute to work correctly. For more information, see **Grid Activation (Section 4.7.3.5.2)**. This attribute allows the cell to become editable when your end user selects it.
- **enableOnF2** - Grid activation must be enabled on your WebGrid for this attribute to work correctly. For more information, see **Grid Activation (Section 4.7.3.5.2)**. This attribute allows the cell to become editable when your end user presses F2.
- **updateMode** - Specifies the update mode of the WebGrid component. You can set the updateMode so that either a single cell is updated or a row is updated.
- **enableOnMouseClick** - Specifies which mouse click initializes editing. You can set the enableOnMouseClick to accept a double or single click. By default editing is initialized when the end user double clicks.
- **rowValueChangeListener** - Specifies which method is called on the server when a row value is changed.
- **cellValueChangeListener** - Specifies which method is called on the server when a single cell value is changed.

The <ig:gridClientEvents> tag is used to combine all properties for client events into one tag.

The following is a list of all the attributes within the <ig:gridClientEvents> tag:

- **enteringEditMode** - Specifies a function to be called when a cell/row is entering in editing mode.
- **enteredEditMode** - Specifies a function to be called when a cell/row is entered in editing mode
- **exitingEditMode** - Specifies a function to be called when a cell/row is exiting from editing mode.
- **exitedEditMode** - Specifies a function to be called when a cell/row is exited from editing mode.

The <ig:column> tag within an editable grid can be set to non-editable using the following attribute:

- **readOnly** - Specifies if the column is editable. Default value is false.

Related Topic

Cell Editing (Section 4.7.3.4.1)

4.7.3.5 Navigating and Selecting

Click the following links to view topics written specifically to help you navigate through your grid and select data from your grid.

- **Change Default Setting of ColumnSelectRow (Section 4.7.3.5.1)**
- **Grid Activation (Section 4.7.3.5.2)**
- **Grid Activation Properties (Section 4.7.3.5.3)**
- **Load On Demand Scrolling (Section 4.7.3.5.4)**
- **Multi-Column Sorting (Section 4.7.3.5.5)**
- **Paging (Section 4.7.3.5.6)**
- **Row Selection (Section 4.7.3.5.7)**
- **SelectedRowsChangeEvent (Section 4.7.3.5.8)**
- **Sorting (Section 4.7.3.5.9)**
- **Sorting a Localized WebGrid (Section 4.7.3.5.10)**

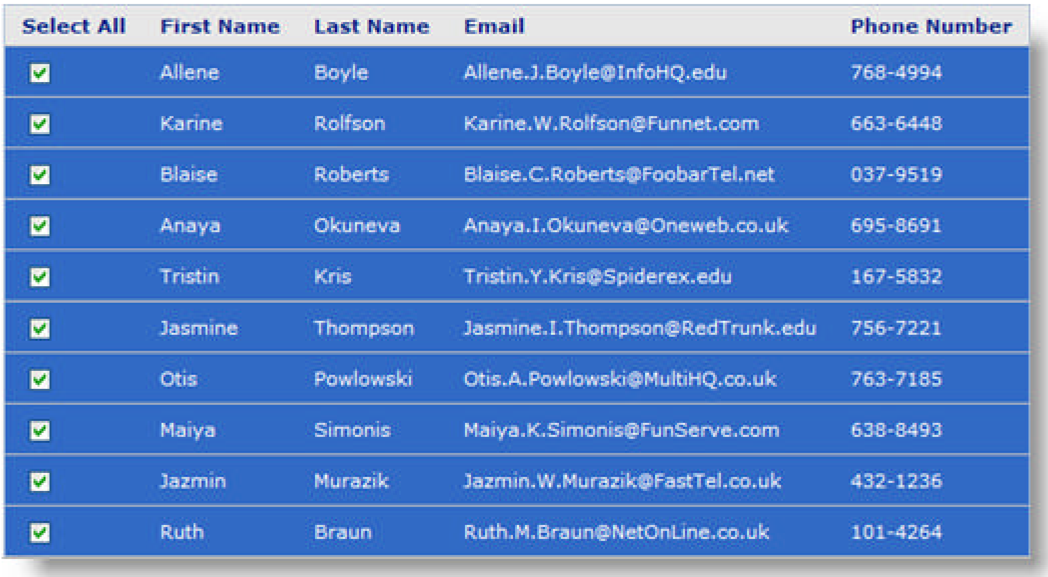
4.7.3.5.1 Change Default Setting of ColumnSelectRow

By default when using the ColumnSelectRow component, each of the checkboxes are unselected when they are initially loaded. There are certain situations where you would want the checkboxes to be selected when they are initially loaded. To do this you simply set the defaultSelected attribute to True in the <ig:columnSelectRow> tag.

The following code demonstrates how to override the default setting of ColumnSelectRow

In JSP:

```
<ig:ColumnSelectRow defaultSelected = "true">
```



Select All	First Name	Last Name	Email	Phone Number
☑	Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
☑	Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
☑	Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
☑	Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
☑	Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
☑	Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221
☑	Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185
☑	Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493
☑	Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236
☑	Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264

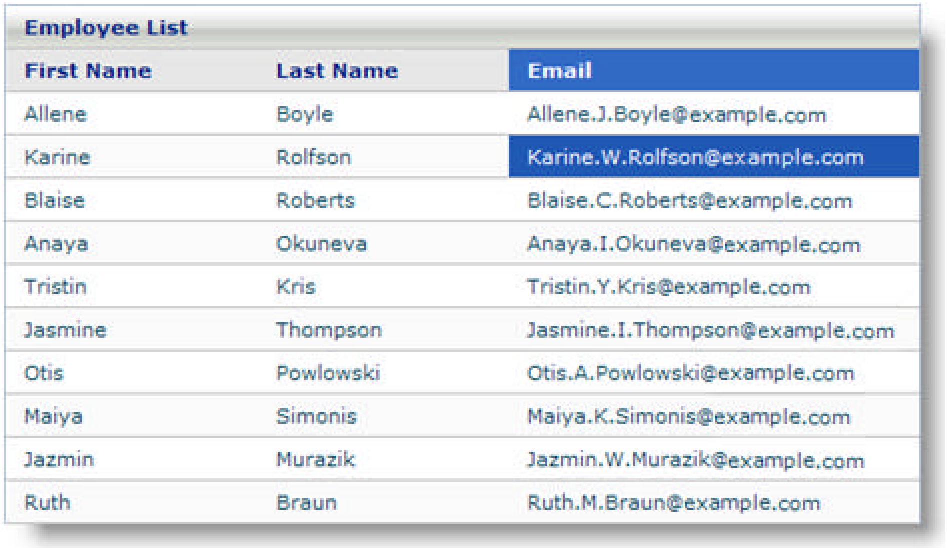
4.7.3.5.2 Grid Activation

The grid activation feature allows your end user to navigate through the cells or rows in your *WebGrid*™ using the keyboard or mouse and activate each cell or row.

By default, activation is disabled on the WebGrid component.

This topic will demonstrate how to enable grid activation which allows your end user to navigate through the cells, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Employee List

First Name	Last Name	Email
Allene	Boyle	Allene.J.Boyle@example.com
Karine	Rolfson	Karine.W.Rolfson@example.com
Blaise	Roberts	Blaise.C.Roberts@example.com
Anaya	Okuneva	Anaya.I.Okuneva@example.com
Tristin	Kris	Tristin.Y.Kris@example.com
Jasmine	Thompson	Jasmine.I.Thompson@example.com
Otis	Powlowski	Otis.A.Powlowski@example.com
Maiya	Simonis	Maiya.K.Simonis@example.com
Jazmin	Murazik	Jazmin.W.Murazik@example.com
Ruth	Braun	Ruth.M.Braun@example.com

Keyboard Support:

- Arrow Keys -- allows the end user to navigate up, down, right or left.
- TAB Key -- allows the end user to navigate to next cell in the row. When the last cell in the current row is reached, pressing the TAB key will activate the first cell in the next row.
- ENTER Key -- allows the end user to navigate to the cell below the activated cell. This is used in combination with Cell Editing. For more information, see **Cell Editing (Section 4.7.3.4.1)**.
- SHIFT + ENTER -- allows the end user to move to the cell above the activated cell. This is used in combination with Cell Editing. For more information, see **Cell Editing (Section 4.7.3.4.1)**.
- SHIFT + TAB -- allows the end user to move to the previously activated cell.

Mouse Support:

- The end user can simply click on a cell or row to activate it.

To enable grid activation, you simply add the <ig:gridActivation> tag to your WebGrid component.

The following code demonstrates how to set activation on your WebGrid.

To enable grid activation using code:

In JSP:

```
<ig:gridView dataSource="#{webgrid_activation.employees}"
    pageSize="10">
    <ig:gridActivation />
    ...
    ...
</ig:gridView>
```

In Java:

```
HtmlGridActivation ga = new HtmlGridActivation();
UIComponentHelper.setFacet(grid.getTemplateItemsBehaviors(),
                           GridView.FACET_GRID_ACTIVATION, ga);
```

To enable grid activation using RAD:

1. From the palette, drag the GridActivation component to your WebGrid.
2. In the Properties window, under ig:gridActivation, you can view and set the properties for GridActivation.

4.7.3.5.3 Grid Activation Properties

This topic describes all the attributes within the <ig:gridActivation> tag.

Setting the <ig:gridActivation> tag allows your end users to navigate through the cells or rows in your *WebGrid*™ using the keyboard or mouse and activate each cell or row. This tag is nested within the <ig:gridView> tag.

The following is a list of all the attributes within the <ig:gridActivation> tag:

- **binding** - This is a standard JSF attribute. Specifies the backing bean you want to wire your components to. This attribute can only use a value expression.
  In JSP:

  ```
  <ig:gridActivation binding="#{bean.uiComponent}"/>
  ```

- **Id** - Specifies the unique id of the component. This attribute doesn't support value expression.
  In JSP:

  ```
  <ig:gridActivation id="gridActivation1"/>
  ```

- **rendered** - This is a standard JSF attribute. Specifies if the component should be displayed. Can be set to True or False. This attribute can also use a value expression.
- **activeCellChangedListener** - This is a Value Change Event. This attribute can only use a value expression.
  In JSP:

  ```
  <ig:gridActivation activeCellChangedListener="{#webGridActivationBean.activeCe
  ```

  In Java:

  ```
  public void activeCellChangedListener(ActiveCellChangedEvent event) {...}
  ```

  The activeCellChangedEvent method accepts ActiveCellChangedEvent as an argument. This event extends ValueChangeEvent.
- **activeCellStyleClass** - Specifies the Cascading Style Sheet (CSS) file to use on an active cell.
  In JSP:

  ```
  <ig:gridActivation activeCellStyleClass="ActiveCellStyle" />
  ```

- **activeColumnStyleClass** - Specifies the CSS file to use on an active column.
  In JSP:

  ```
  <ig:gridActivation activeColumnStyleClass="ActiveColumnStyle" />
  ```

- **activeRowStyleClass** - Specifies the CSS file to use on an active row. You must set <ig:columnSelectRow /> for this attribute to work.
  In JSP:

  ```
  <ig:gridActivation activeRowStyleClass="ActiveRowStyle" />
  ```

Related Topic

Grid Activation (Section 4.7.3.5.2)

4.7.3.5.4 Load On Demand Scrolling

The WebGrid component features load-on-demand to empower users requesting large volumes of data on an only as-requested basis. This protocol allows pages to load much more quickly as they grab only the data that is required.

> **Note:** You must set the height attribute of the <gridView> tag, for more information see **Vertical Scroll Bars (Section 4.7.3.3.6)**, so that a vertical scroll bar is displayed, allowing the end user to scroll

To enable load-on-demand within your WebGrid, you simply set the loadOnDemand attribute within the <gridView> tag.

Setting the rowFetchSize attribute within the <gridView> tag, allows you to specify how many rows to retrieve from the server on each load on demand.

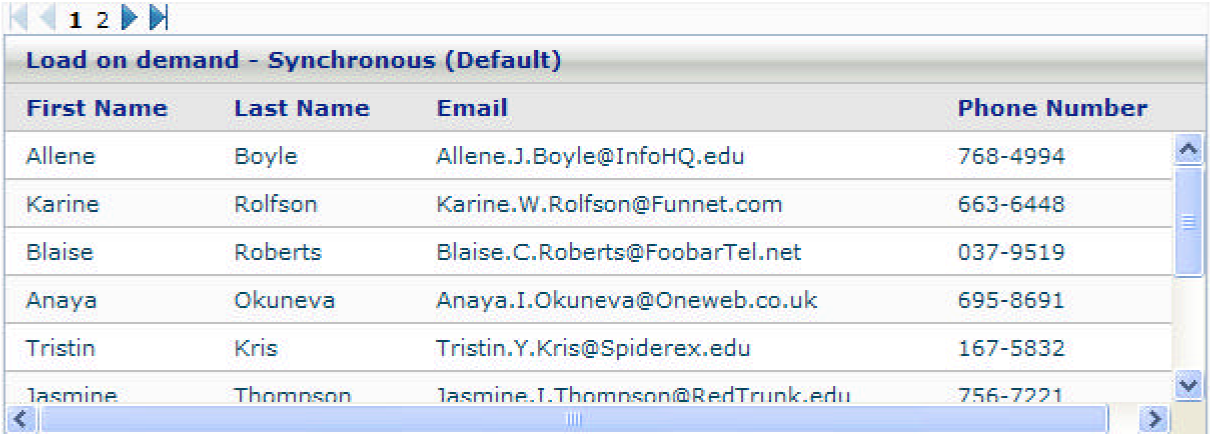
You can set the loadOnDemand attribute to the following values:

- **default** -- In this mode, the WebGrid component will initially load the number of rows specified by the rowFetchSize attribute. When the end user scrolls to the end of the visible grid section, a postback is issued to the server, which will in turn send the number of rows specified by the rowFetchSize attribute to the client and these rows will be appended to the rows on the client.
- **partitioned** -- In this mode, the WebGrid component will initially load the number of rows specified by the rowFetchSize attribute and the vertical scroll bar is set so that it visually accomodates all rows in the underlying data set. WebGrid is divided into virtual pages, whenever the end user scrolls to any section of the WebGrid component, it is calculated weather the virtual page has changed. If it is changed, the number of rows specified by the rowFetchSize attribute is retrieved from the server and displayed in the visible body of WebGrid.

> **Note:** When you are using load on demand, you must ensure that the total height of the number of rows specified by the rowFetchSize attribute is larger than the CSS height that is set in the style attribute for your WebGrid.

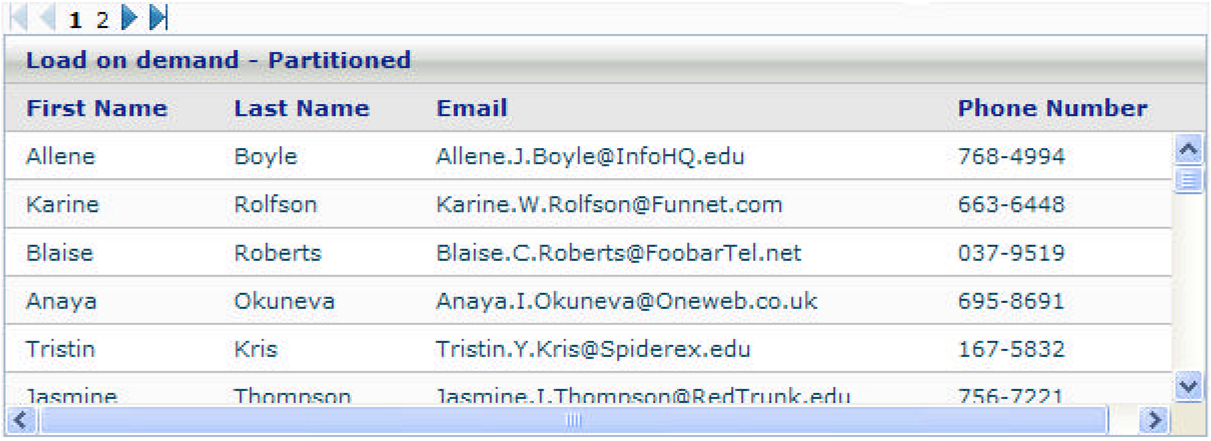
This topic will demonstrate how to enable load-on-demand scrolling using default mode and partitioned mode settings, as illustrated in the screen shots below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Load on demand - Synchronous (Default)

First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.J.Thompson@RedTrunk.edu	756-7221



Load on demand - Partitioned

First Name	Last Name	Email	Phone Number
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221

To enable load-on-demand scrolling using code:

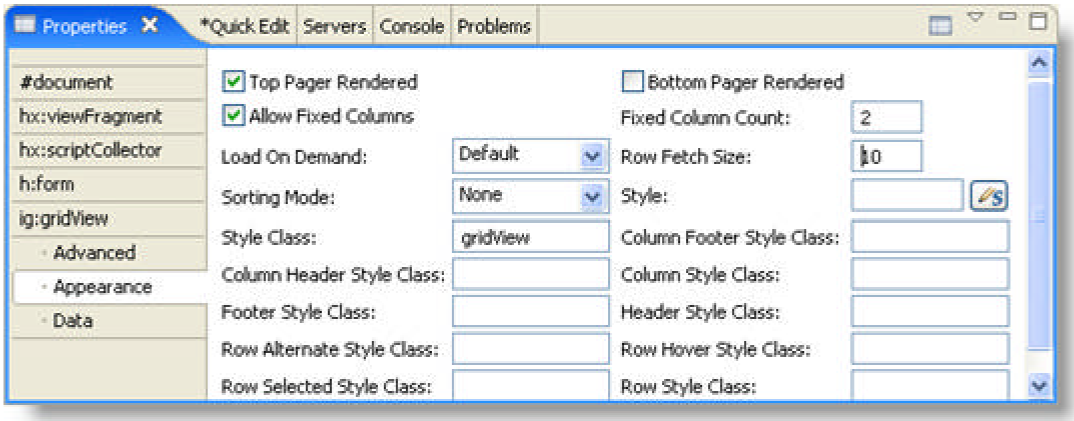
1. Set the following attributes of the <ig:gridView> tag, as shown in the JSP code below
   - loadOnDemand -- default
   - rowFetchSize -- 10

   In JSP:

```
<ig:gridView
    dataSource="#{webgrid_automaticPagingPage.employees}"
    loadOnDemand="default"
    rowFetchSize="10"
    pageSize ="40"
    style="height: 200px; width: 600px; margin-bottom: 40px" />
```

To enable load-on-demand using RAD:

1. In Design view, select the WebGrid component.
2. In the Properties window, under ig:gridView, select Appearance.
3. Set the following properties:
   - Load On Demand -- Default
   - Row Fetch Size -- 10



☑ Top Pager Rendered

□ Bottom Pager Rendered

☑ Allow Fixed Columns

Fixed Column Count: 2

Load On Demand: Default

Row Fetch Size: 10

Sorting Mode: None

Style:

Style Class: gridView

Column Footer Style Class:

Column Header Style Class:

Column Style Class:

Footer Style Class:

Header Style Class:

Row Alternate Style Class:

Row Hover Style Class:

Row Selected Style Class:

Row Style Class:

NetAdvantage for JSF

4.7.3.5.5 Multi-Column Sorting

WebGrid™ allows you to enable the sorting of multiple columns. The end user can accomplish this functionality by clicking on the first column, holding the SHIFT key, and then clicking on the second column to be sorted.

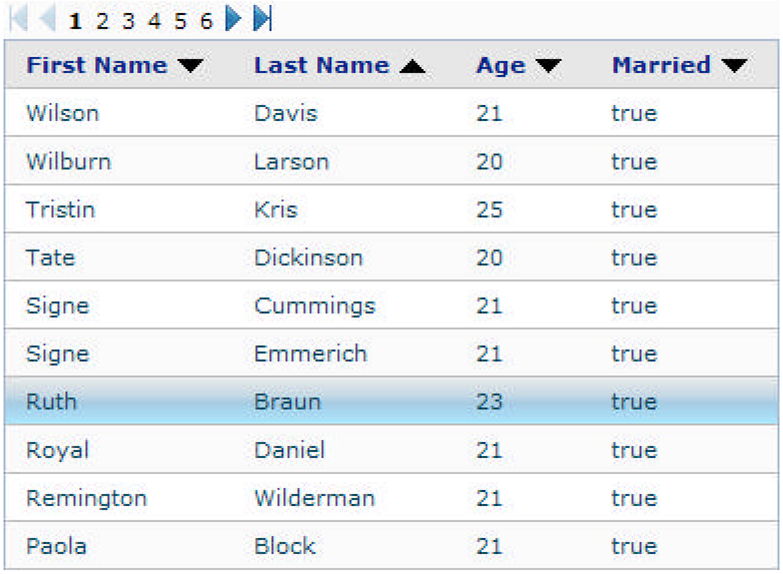
Multiple-column sorting works by allowing the first column that is sorted to take the highest precedence over the sorting of other columns.

Enabling multi-column sorting on your WebGrid allows your end users to view their WebGrid data in the order that they choose, giving them more control over the layout and readability of your WebGrid.

The WebGrid component automatically perform AJAX-based multi-column sorting, and refresh only the grid portion of the Web page by using our unique Smart Refresh™ and Smart Databinding technology.

This topic will demonstrate how to enable multi-column sorting, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name ▼	Last Name ▲	Age ▼	Married ▼
Wilson	Davis	21	true
Wilburn	Larson	20	true
Tristin	Kris	25	true
Tate	Dickinson	20	true
Signe	Cummings	21	true
Signe	Emmerich	21	true
Ruth	Braun	23	true
Royal	Daniel	21	true
Remington	Wilderman	21	true
Paola	Block	21	true

To enable multi-column sorting using code:

1. Set the sortingMode attribute of the <ig:gridView> tag to multi, as shown in the JSP code below.
   In JSP:

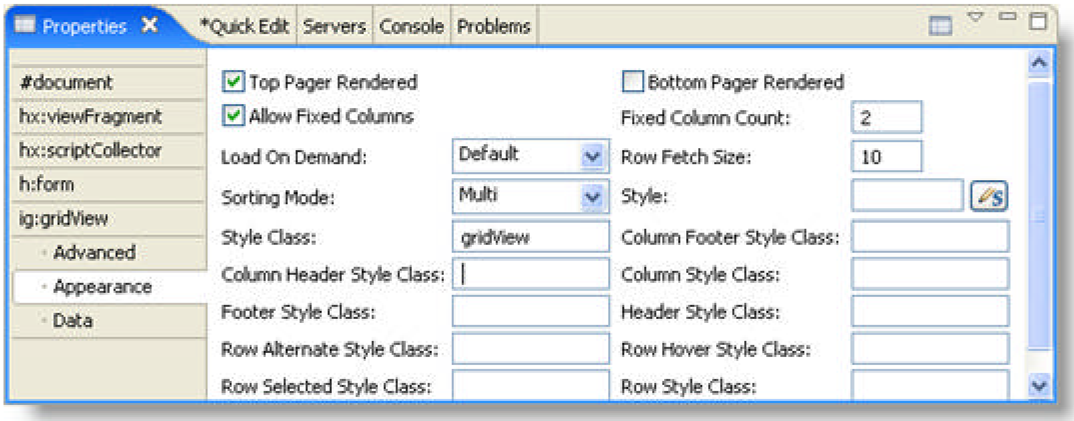
```
<ig:gridView
    dataSource="#{webgrid_multicolumnSortingPage.employees}"
    pageSize="10"
    sortingMode="multi"
    sortListener="#{webgrid_multicolumnSortingPage.sortListener}">
```

2. Set the sortBy attribute of the <ig:column> tag, as shown in the JSP code below.
   In JSP:

```
<ig:column
    sortBy ="firstName">
```

To enable multi-column sorting using RAD:

1. In Design view, select the WebGrid component.
2. In the Properties window, under ig:gridView, select Appearance.
3. Set the Sorting Mode property to Multi.



☑ Top Pager Rendered

☐ Bottom Pager Rendered

☑ Allow Fixed Columns

Fixed Column Count: 2

Load On Demand: Default

Row Fetch Size: 10

Sorting Mode: Multi

Style:

Style Class: gridView

Column Footer Style Class:

Column Header Style Class:

Column Style Class:

Footer Style Class:

Header Style Class:

Row Alternate Style Class:

Row Hover Style Class:

Row Selected Style Class:

Row Style Class:

4.7.3.5.6 Paging

With WebGrid™, you can take advantage of its paging feature. By using paging, you allow your end users to view a specified number of rows in the grid, instead of all the data at once.

The paging functionality automatically performs AJAX-based paging, and refreshes only the grid portion of the Web page by using our unique Smart Refresh™ and Smart Databinding technology.

The pager component allows you to navigate through your grid:

- using a First, Last, Previous and Last pattern.
- choosing a specific page.

> **Note:** If the rows of your grid are less than the number of rows specified for each page, then the pager component will not appear.

You can navigate through your grid using Automatic Paging or Customizable Paging.

Automatic Paging

You can configure the WebGrid components to automatically display and handle the pager component, which allows you to directly move to specific page indexes. To enable automatic paging, simply set the pageSize attribute of the <gridview> tag.

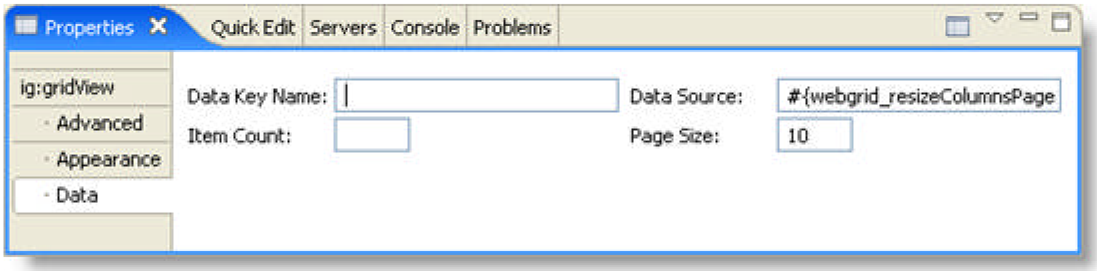
To enable automatic paging:

1. Set the <pageSize> attribute of the gridView tag:
   In JSP:

```
<ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
     pageSize="10"
     fixedColumnCount="2"
     resizableColumns="true"
     style= "width: 600px">
```

To enable automatic paging in IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. In Design view, select the grid component on your page.
2. In the Properties tab, under ig:gridView, select Data.
3. In the Page Size box, type 10.



Data Key Name:
Item Count:
Data Source: #{webgrid_resizeColumnsPage
Page Size: 10

Customizable Paging

You can customize the display of the pager component on your grid. The following options are available:

- Display First and Last Page
- Display Previous and Next Page
- Display Page Indexes

You can also set:

- the number of rows per page
- maximum number of pages to display in the index

You can use the <ig:pager/> tag to customize your pager component.

To customize the pager component:

1. Set the <bottomPagerRendered> attribute to True within the <ig:gridView> tag to display the pager at the bottom of the grid.
   Set the <topPagerRendered> attribute to True within the <ig:gridView> tag to display the pager at the top of the grid.
   In JSP:

   ```
   <ig:gridView dataSource="#webgrid_resizeColumnsPage.employees}"
        pageSize="10"
        bottomPagerRendered="true"
        topPagerRendered="true">
   ```

2. Create a facet named "topPager" or "bottomPager" within the <ig:gridView> tag.
3. Create an <ig:pager> tag.
   In JSP:

   ```
   <f:facet name="topPager">
       <ig:pager />
    </f:facet>
   ```

4. Set the attributes for the <ig:pager> tag.
   The following attributes can be set:
   - firstLastRendered -- Specifies whether or not the first and last pages are visible.
   - maximumPageNumders -- Specifies the maximum number of pages to display.
   - pageNumberHoverStyleClass -- Specifies the CSS class applied to a page number when the mouse pointer hovers over it.
   - pageNumberSelectedStyleClass -- Specifies the CSS class applied to a selected page number.
   - pageNumbersRendered -- Specifies whether or not page numbers are visible.
   - pageNumberStyleClass -- Specifies the CSS class applied to the page numbers.
   - previousNextRendered -- Specifies whether or not the previous and next pages are visible.
   - style -- Specifies the style for the pager.
   - styleClass -- Specifies the CSS class applied to the pager.

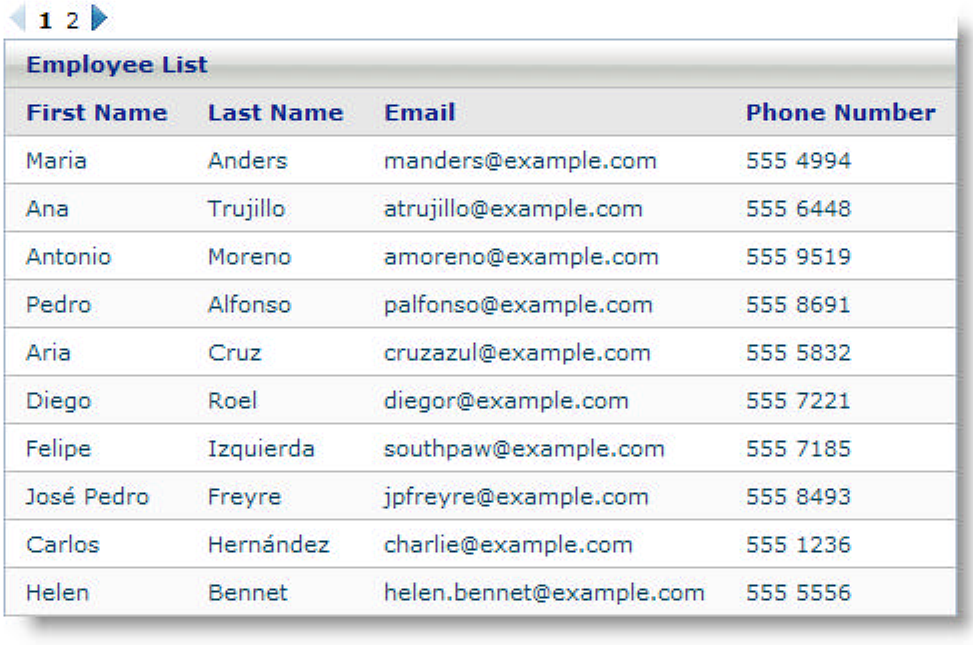
   The following example code sets the maximum number of pages to display to two and does not display the first and last pages.

   In JSP:

   ```
   <ig:pager maximumPageNumbers="2"
   ```

```
        firstLastRendered="false"/>
```

The following screen shot shows a grid with a customized pager.



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

4.7.3.5.7 Row Selection

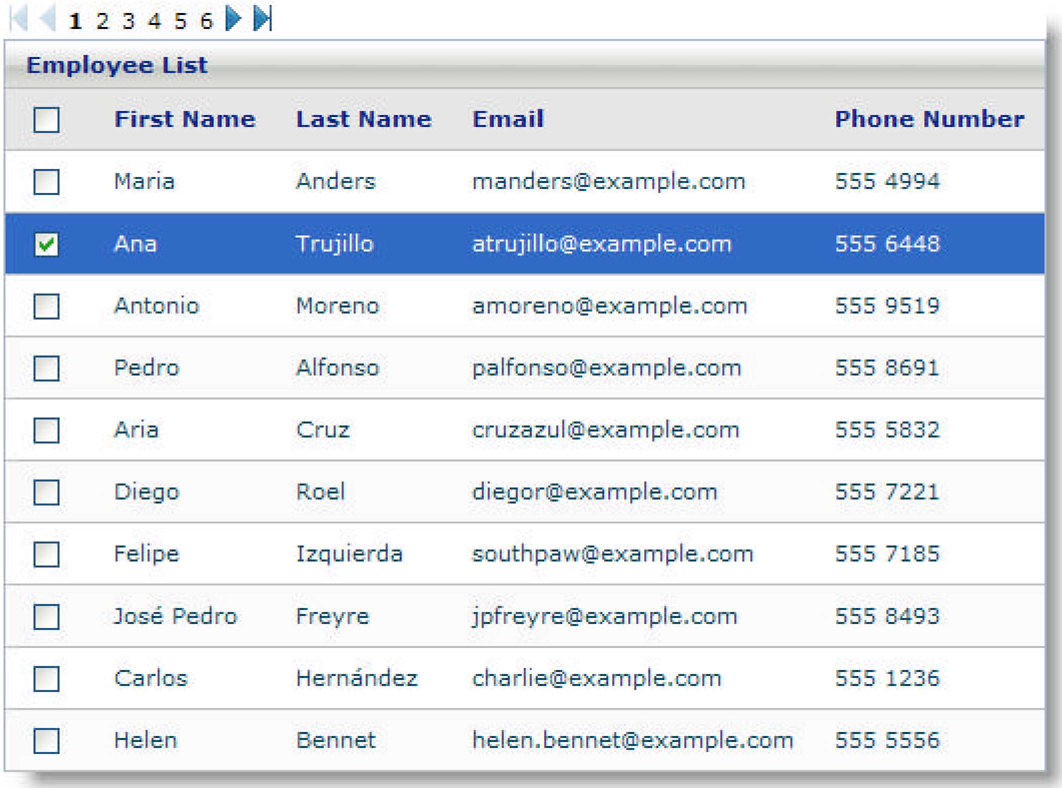
You can allow your end users to select a row in WebGrid™ by using a Row Selection Event and Row Selection Listener, allowing the end user to interact with the grid and perform an action based on their selection or deselection actions.

For example, if an end user selects a row, the information from that row selection can be displayed elsewhere on the form.

> **Note:** The RowSelection event fires when rows are selected or deselected. The event sends the SelectedRowsChangedEvent arguments to the backing bean method. The passed in argument has a method called getSelectedRowsChange, which is a list of type SelectedRowChange, and contains only the newly selected or deselected rows (previously selected or deselected rows will not be in this list). Therefore, you would need to loop over the list to figure out which rows were selected versus deselected, each time.

This topic will demonstrate how to enable row selection, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Employee List

□	First Name	Last Name	Email	Phone Number
□	Maria	Anders	manders@example.com	555 4994
☑	Ana	Trujillo	atrujillo@example.com	555 6448
□	Antonio	Moreno	amoreno@example.com	555 9519
□	Pedro	Alfonso	palfonso@example.com	555 8691
□	Aria	Cruz	cruzazul@example.com	555 5832
□	Diego	Roel	diegor@example.com	555 7221
□	Felipe	Izquierda	southpaw@example.com	555 7185
□	José Pedro	Freyre	jpfreyre@example.com	555 8493
□	Carlos	Hernández	charlie@example.com	555 1236
□	Helen	Bennet	helen.bennet@example.com	555 5556

> **Note:** This topic assumes that you have a grid created. For information on how to create a grid, see **Getting Started with WebGrid (Section 4.7.2)**.

To enable row selection using code:

1. Bind the <selectedRowsChangeListener> attribute of the gridView tag to a method in the backing bean
   In JSP:

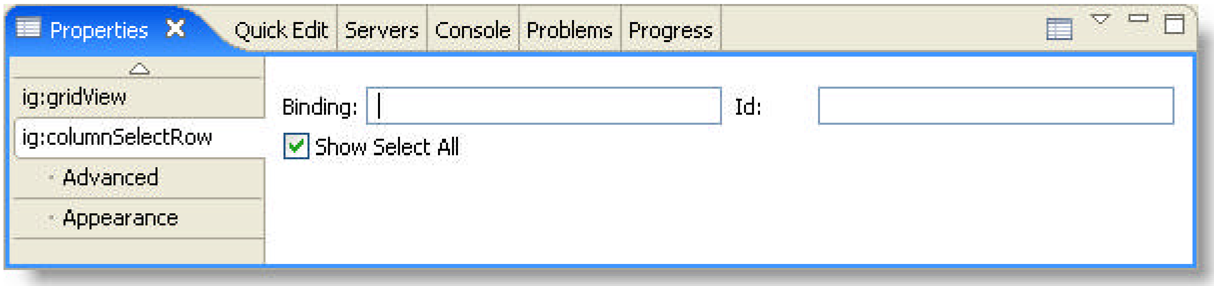
```
<ig:gridView
  binding="#{webgrid_rowSelectionPage.grid}"
  dataKeyName="id"
  dataSource="#{webgrid_rowSelectionPage.employees}"
  pageSize="10"
  selectedRowsChangeListener="#{webgrid_rowSelectionPage.onSelectedRowsChange}" >
```

2. Create a <columnSelectRow /> tag and set the showSelectAll attribute to True.
   In JSP:

```
<ig:columnSelectRow showSelectAll="true" />
```

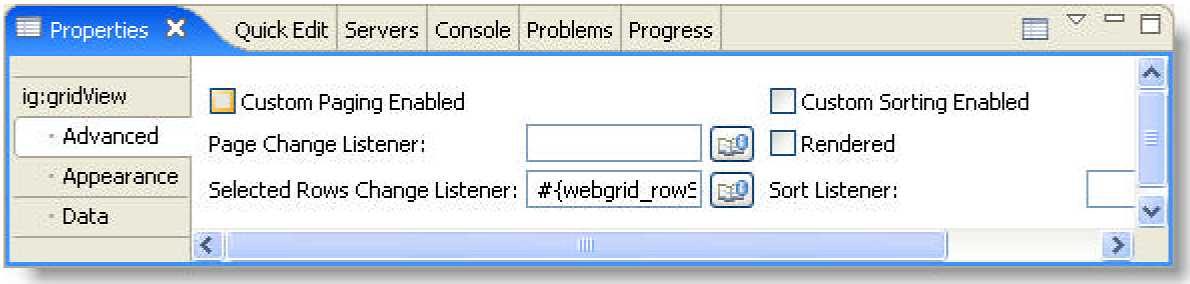
To enable row selection in RAD:

1. From the palette drag the ColumnSelectRow component on to your grid.
2. In Design view, select the ColumnSelectRow component on your page.
3. Select the Show Select All check box.



Binding:
Id:
☑ Show Select All

4. In the Properties tab, under the ig:gridView, select Advanced.
5. In the Selected Rows Change Listener, type {webgrid_rowSelectionPage.onSelectedRowsChange}.



□ Custom Paging Enabled
□ Custom Sorting Enabled
Page Change Listener:
□ Rendered
Selected Rows Change Listener: #{webgrid_rowS
Sort Listener:

4.7.3.5.8 SelectedRowsChangeEvent

The SelectedRowsChangeEvent event now contains two additional methods, isSelectAll() and selectAllIsChecked (). With these methods you can determine if the SelectAll checkbox is selected and also if the event was triggered from the SelectAll checkbox or from select row checkbox.

- isSelectAll() – returns true if the state of the SelectAll checkbox has changed state, else returns false.
- selectAllIsChecked() – returns true is the SelectAll checkbox is selected, else returns false.

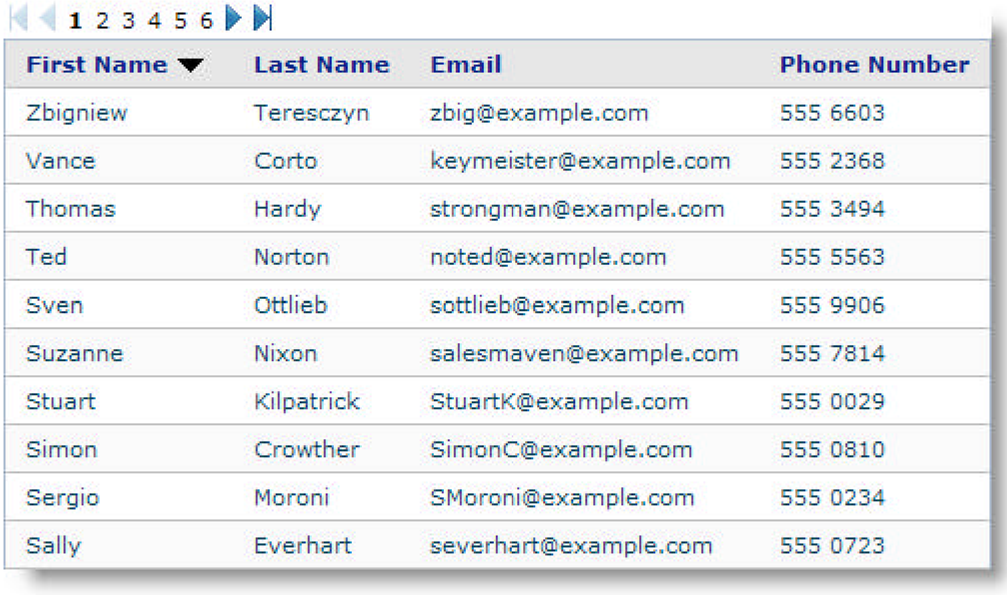
4.7.3.5.9 Sorting

Automatic sorting is an important piece of functionality provided by the WebGrid™ components. WebGrid automatically displays and handles the sorting for you. Your end user can sort on the columns by clicking the column headers, allowing them to view the grid data in the order that they want. Enabling sorting on your grid allows your end users to view their grid data in the order that they choose, giving them more control over the layout and readability of your grid.

The WebGrid components automatically perform AJAX-based sorting, and refresh only the grid portion of the Web page by using our unique Smart Refresh™ and Smart Databinding technology.

This topic will demonstrate how to enable sorting, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name ▼	Last Name	Email	Phone Number
Zbigniew	Teresczyn	zbig@example.com	555 6603
Vance	Corto	keymeister@example.com	555 2368
Thomas	Hardy	strongman@example.com	555 3494
Ted	Norton	noted@example.com	555 5563
Sven	Ottlieb	sottlieb@example.com	555 9906
Suzanne	Nixon	salesmaven@example.com	555 7814
Stuart	Kilpatrick	StuartK@example.com	555 0029
Simon	Crowther	SimonC@example.com	555 0810
Sergio	Moroni	SMoroni@example.com	555 0234
Sally	Everhart	severhart@example.com	555 0723

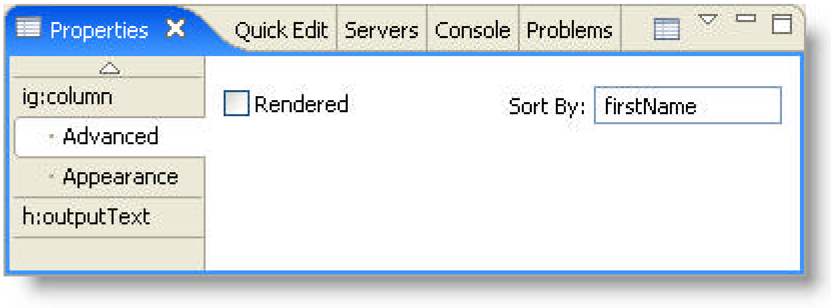
To enable sorting using code:

1. Set the "sortBy" attribute of the <ig:column> tag, as shown in the JSP code below.
   In JSP:

```
<ig:column sortBy="firstName">
       <f:facet name="header">
           <h:outputText value="First Name" />
       </f:facet>
       <h:outputText value="#{DATA_ROW.firstName}" />
    </ig:column>
```

To enable sorting using RAD:

1. In Design view, select the column on which you want to enable sorting.
2. In the Properties window, under ig:column, select Advanced.
3. In the Sort By box, type firstName.



□ Rendered

Sort By: firstName

4.7.3.5.10 Sorting a Localized WebGrid

By setting the newly added attributes language, country and variant in the tag, you can sort the data based on different languages. For example if your data was in German, you can now set the sorting based on the German alphabet so that German characters such as the Umlaut is sorted correctly.

The language attribute accepts a two letter code as defined by **ISO639.2 (http://www.loc.gov/standards/iso639-2/php/English_list.php)**

The country attribute accepts a two letter code as defined by **ISO 3166 (http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.h**

The variant attribute is a browser specific code defined by the **Java Locale API (http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html)**.

In JSP:

```
<ig:gridView language=”de” variant=""WIN"" country="de">
```

4.7.4 WebGrid Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.8 WebInput

Click the links below to find information specific to the *WebInput*™ components.

- **About the WebInput Components (Section 4.8.1)** -- Provides information about the key functionalities of the WebInput components.
- **WebInput Accessibility Compliance (Section 4.8.2)** -- This topic outlines the way in which WebInput meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebInput control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.8.1 About the WebInput Components

The following is a list of the *WebInput*™ components:

- **CheckBox (Section 4.8.1.1)**
- **CheckboxList (Section 4.8.1.2)**
- **DateChooser (Section 4.8.1.3)**
- **DropdownList (Section 4.8.1.4)**
- **InputCurrency (Section 4.8.1.5)**
- **InputDecimal (Section 4.8.1.6)**
- **InputEmail (Section 4.8.1.7)**
- **InputPercent (Section 4.8.1.8)**
- **InputRegularExpression (Section 4.8.1.9)**
- **RadioButton (Section 4.8.1.10)**
- **RadioButtonList (Section 4.8.1.11)**

The WebInput components allow you to display various components in your application to enable the user to input various types of data (e.g., dates, e-mail addresses, regular expressions) and make selections using check boxes, radio buttons, or drop-down lists.

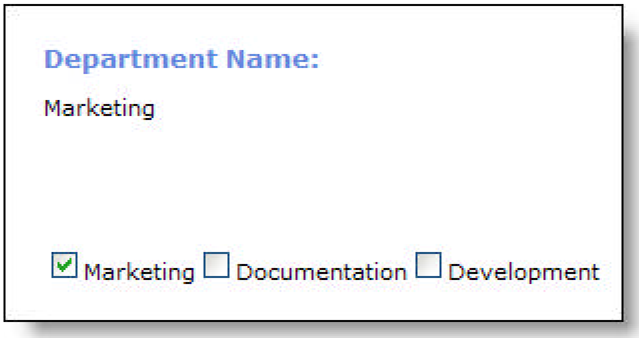
4.8.1.1 CheckBox

The CheckBox component is used for selection in your application and has the ability to cause other components to be updated when their values change.

AJAX is used for updating when a check box is selected.

This topic will demonstrate how to use the CheckBox component and display a value when the check box is selected, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**


Department Name:

Marketing

☑ Marketing □ Documentation □ Development

To create a CheckBox component using code:

1. Open the <ig:checkbox> tag and set the following attributes, as shown in the JSP code below:
   - binding -- #{webinput_checkbox.checkbox1}
   - label -- Marketing
   - smartRefreshIds -- outputSelectedCheckbox
   - valueChangeListener -- #{webinput_checkBox.onValueChange}

   In JSP:

```
<ig:checkBox
    binding="#{webinput_checkBox.checkbox1}"
    label="Marketing"
    smartRefreshIds="outputSelectedCheckbox"
    valueChangeListener="#{webinput_checkBox.onValueChange}"/>
```

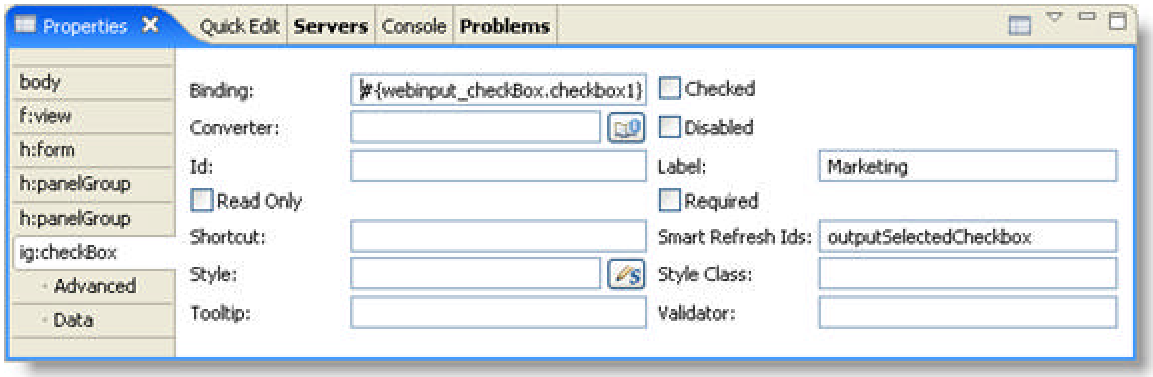
2. Repeat for each check box.
3. Open the <h:outputText> tag and set the following attributes as shown in the JSP code below. This is where the values of the check boxes will be displayed:
   - id -- outputSelectedCheckbox
   - value -- #{webinput_checkBox.selectedCheckbox}

   In JSP:

```
<h:outputText
    id="outputSelectedCheckbox"
    value="#{webinput_checkBox.selectedCheckbox}" />
```

To create a CheckBox component using RAD:

1. From the palette, drag the CheckBox component to your page.
2. In the Properties window, select ig:checkbox.
3. Set the following properties:
    - binding -- #{webinput_checkbox.checkbox1}
    - label -- Marketing
    - smartRefreshIds -- outputSelectedCheckbox



Binding: #{webinput_checkBox.checkbox1}

□ Checked

□ Disabled

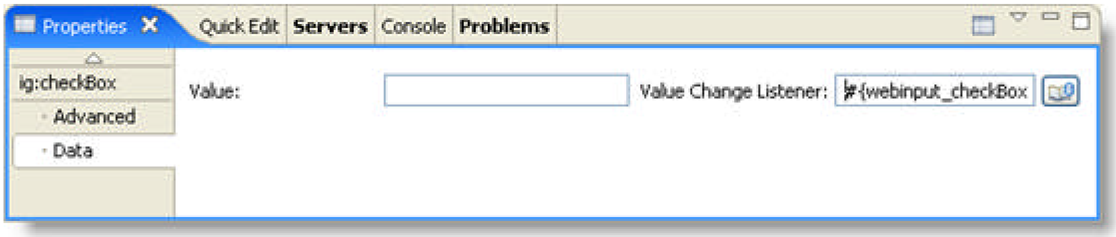
Label: Marketing

□ Read Only

□ Required

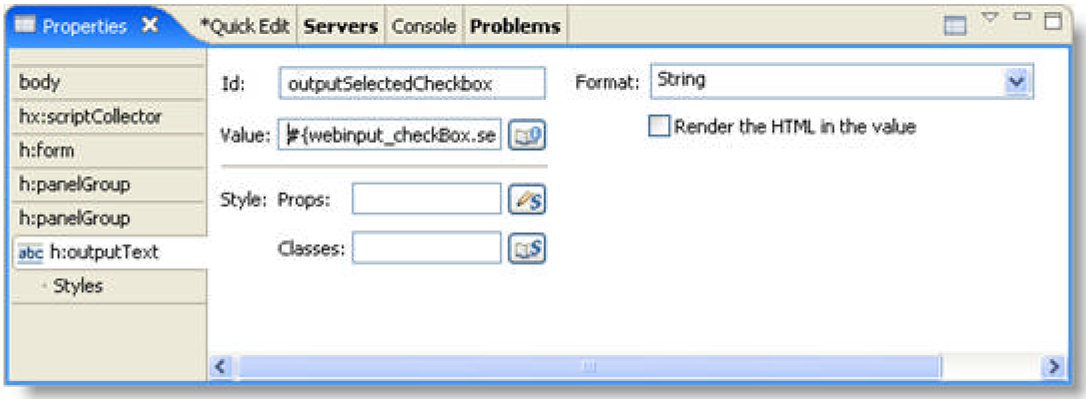
Smart Refresh Ids: outputSelectedCheckbox

4. In the Properties tab, under ig:checkbox, select Data.
5. In the Value Change Listener, set the value to #{webinput_checkBox.onValueChange}.



Value Change Listener: #{webinput_checkBox

6. Repeat for each check box.
7. From the palette, under the Enhanced Faces Components, drag the Output component on to your page.
8. In the Properties window, select h:outputText.
9. Set the following properties:
    Id -- outputSelectedCheckbox
    Value -- #{webinput_checkBox.selectedCheckbox}
    Format -- String



Id: outputSelectedCheckbox

Value: #{webinput_checkBox.se

Format: String

□ Render the HTML in the value

Style: Props:

Classes:

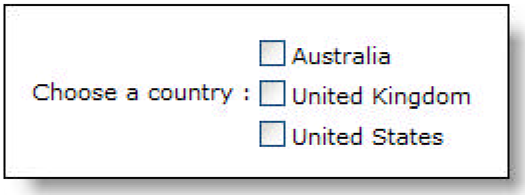
4.8.1.2 CheckboxList

The CheckboxList component is used for selection purposes in your application and can also be configured to update other components based on the values selected. The CheckboxList component is bound to values in your backing bean.

AJAX is used for updating when a check box is selected.

This topic will demonstrate how to create a CheckboxList component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Choose a country : □ Australia
□ United Kingdom
□ United States

To create a CheckboxList component using code:

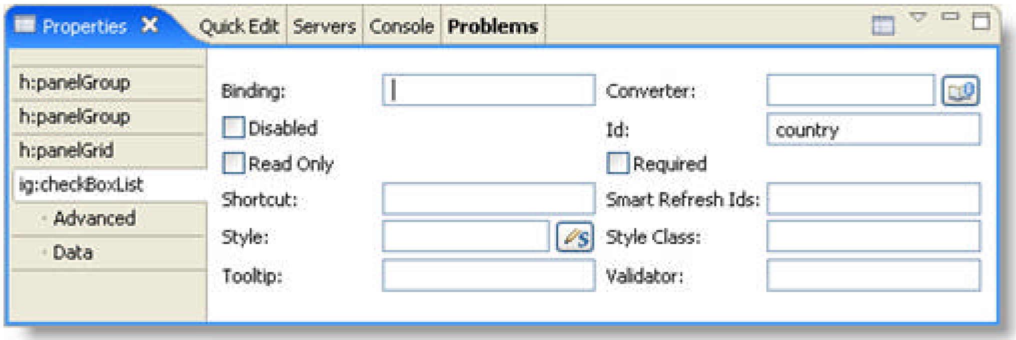
1. Open the <ig:checkBoxList> tag and set the following attributes, as shown in the JSP code below:
    - id -- country
    - dataSource -- #{webinput_checkBoxList.countryList}

    In JSP:

```
<ig:checkBoxList
    id = "country"
    dataSource ="#{webinput_checkBoxList.countryList}"/>
```

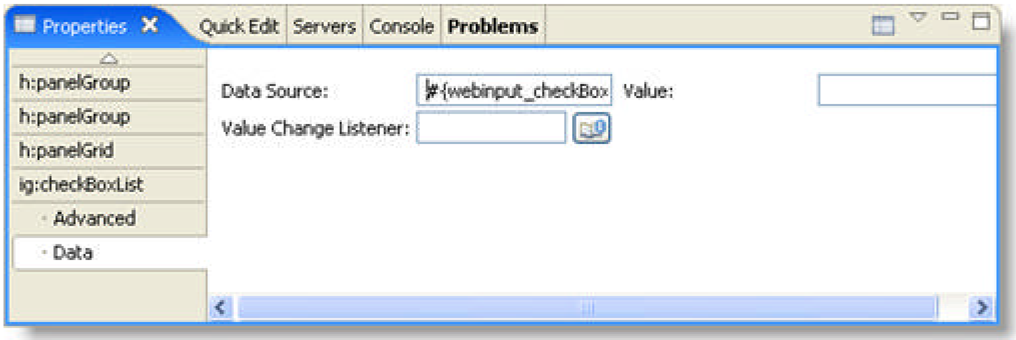
To create a CheckboxList component using RAD:

1. From the palette, drag the CheckboxList component to your page.
2. In the Properties window, select ig:checkBoxList.
3. Set Id to country



Binding:
□ Disabled
□ Read Only
Shortcut:
Style:
Tooltip:
Converter:
Id: country
□ Required
Smart Refresh Ids:
Style Class:
Validator:

4. In the Properties tab, under ig:checkBoxList, select Data.
5. Set Data Source to #{webinput_checkBoxList.countryList}



Data Source: #{webinput_checkBox

Value:

Value Change Listener:

4.8.1.3 DateChooser

The DateChooser component combines a date-specific masked editing field with the drop down support of a combo box. Users can enter dates directly in the control, or click the drop down button to display a calendar for date selection.

The DateChooser component allows you to:

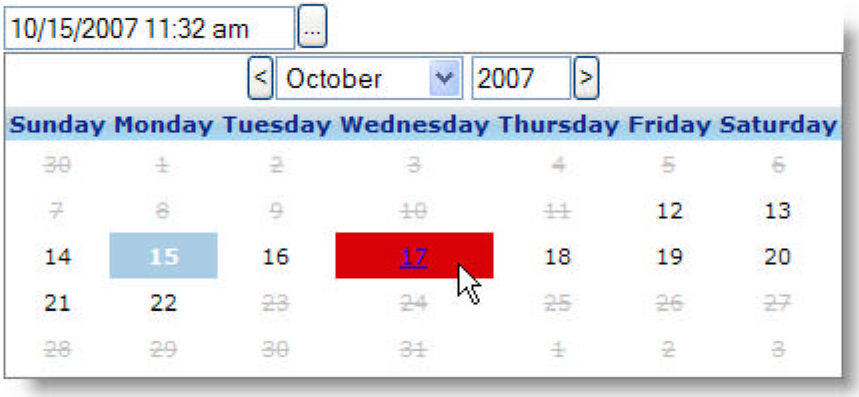
- set minimum and maximum limits on the date range that your end users may enter in to the combo box or select from the calendar.
- change the appearance and style of the DateChooser component.
- specify the day format.
- specify the first day of the week to display.

The DateChooser component can also be used inside of WebGrid to provide an intuitive editing and selection interface for grid columns that handle dates.

> **Note**: When using the DateChooser converter, you need to extend com.infragistics.faces.input.converter.DateTimeConverter

This topic will demonstrate how to create the DataChooser component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



10/15/2007 11:32 am

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
~~30~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~
~~7~~	~~8~~	~~9~~	~~10~~	~~11~~	12	13
14	15	16	17	18	19	20
21	22	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~
~~28~~	~~29~~	~~30~~	~~31~~	~~1~~	~~2~~	~~3~~

To create a DateChooser component using code:

1. Open the <ig:dateChooser> tag and set the following attributes, as shown in the JSP code below:
   - dayHeaderFormat -- Long
   - firstDayOfWeek -- 1
   - editMasks -- MM/dd/yyyy hh:mm a
   - maximumDate -- 10/22/2007
   - minimumDate -- 10/12/2007
   - dayHoverStyleClass -- red

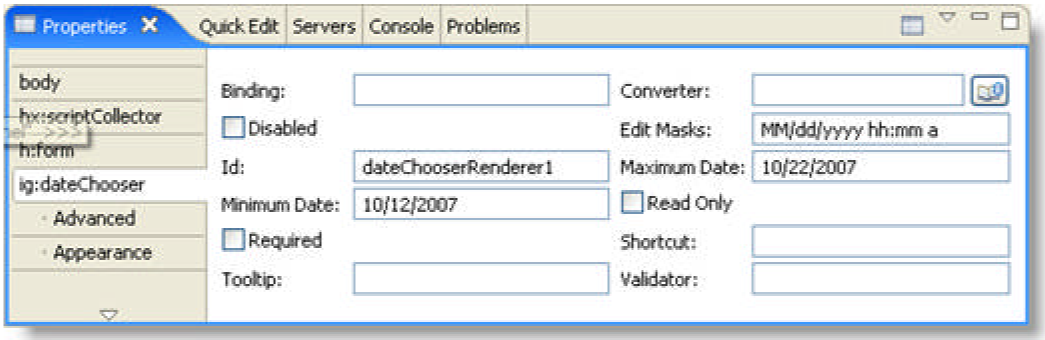
   In JSP:

```
<ig:dateChooser
    dayHeaderFormat ="Long"
```

```
firstDayOfWeek="1"
editMasks="MM/dd/yyyy hh:mm a"
maximumDate="10/22/2007"
minimumDate="10/12/2007"
dayHoverStyleclass="red"/>
```

To create a DateChooser component using RAD:

1. From the palette, drag the DateChooser component to your page.
2. In the Properties window, select ig:dateChooser.
3. Set the following properties:
   - Edit Masks -- MM/dd/yyyy hh:mm a
   - Maximum Date -- 10/22/2007
   - Minimum Date -- 10/12/2007



Binding:

□ Disabled

Id: dateChooserRenderer1

Minimum Date: 10/12/2007

□ Required

Tooltip:

Converter:

Edit Masks: MM/dd/yyyy hh:mm a

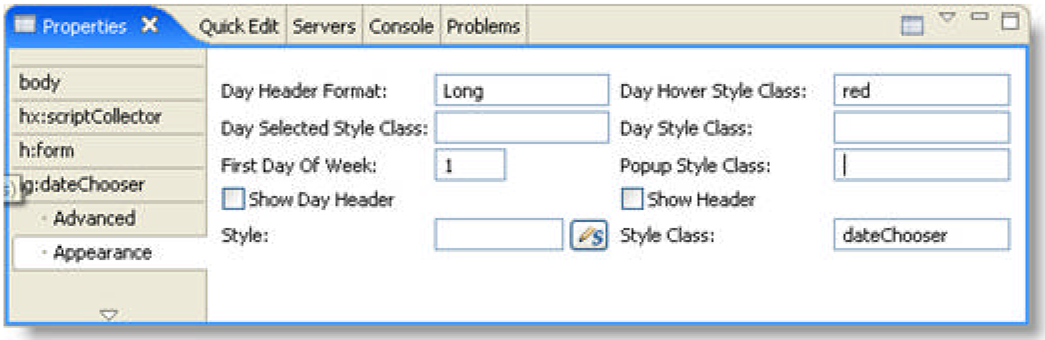
Maximum Date: 10/22/2007

□ Read Only

Shortcut:

Validator:

4. In the Properties tab, under ig:checkbox, select Appearance.
5. Set the following properties:
   - Day Header Format -- Long
   - Day Hover Style Class -- red
   - First Day Of Week -- 1



Day Header Format: Long

Day Selected Style Class:

First Day Of Week: 1

□ Show Day Header

Style:

Day Hover Style Class: red

Day Style Class:

Popup Style Class:

□ Show Header

Style Class: dateChooser

Related Topic

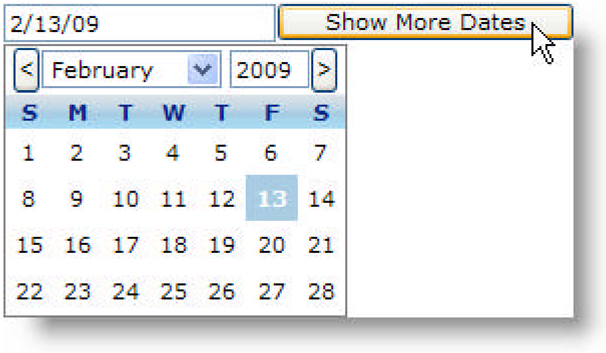
Set the Text of DateChooser Submit Button (Section 4.8.1.3.1)

4.8.1.3.1 Set the Text of the Submit Button

There may be certain situations where the default setting on the DateChooser's submit button is not suitable for your web application. You may require more information to be displayed to your end user that explicitly states what the button does.

This topic will demonstrate how to set the text of the button as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To set the button's value using code:

In JSP:

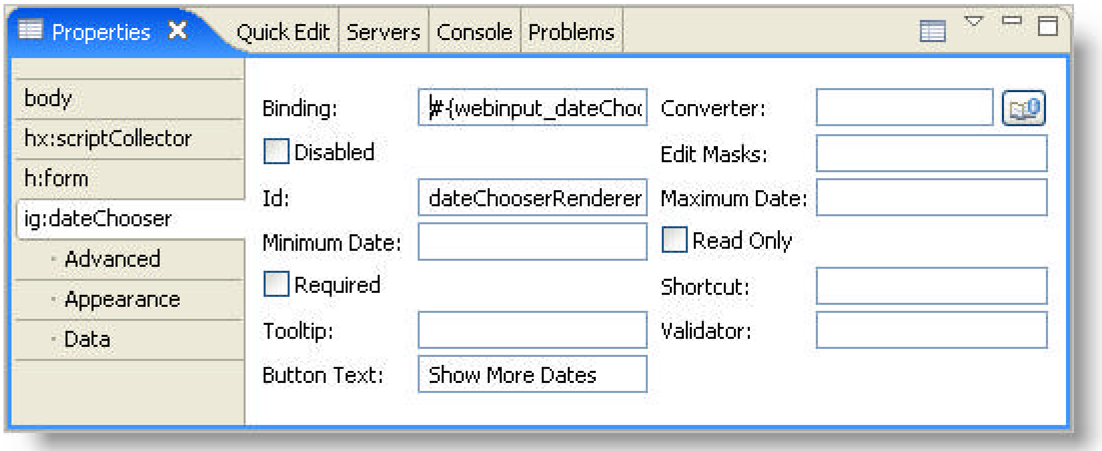
```
<ig:dateChooser buttonText="Show More Date"/>
```

In Java:

```
HTMLDateChooser dateChooser = new HTMLDateChooser();
dateChooser.setButtonText = "Show More Dates";
```

To set the button's value using RAD:

1. From the Palette, drag the DateChooser component to your page.
2. In the Design View, select the DateChooser component.
3. In the properties window, select ig:dateChooser.
4. Set the Button Text property to Show More Dates.



Binding: #{webinput_dateChoc

□ Disabled

Id: dateChooserRenderer

Minimum Date:

□ Required

Tooltip:

Button Text: Show More Dates

Converter:

Edit Masks:

Maximum Date:

□ Read Only

Shortcut:

Validator:

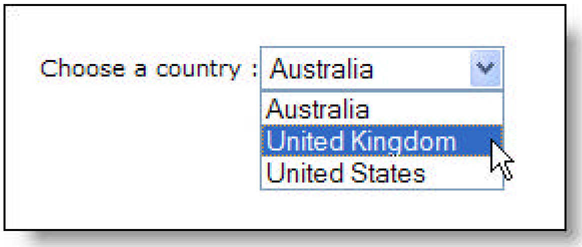
4.8.1.4 DropdownList

The DropdownList component is used for selection purposes in your application and can also be configured to update other components based on the values selected.

AJAX is used for updating when a check box is selected.

This topic will demonstrate how to create a DropdownList component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Choose a country : Australia

To create a DropDownList component using code:

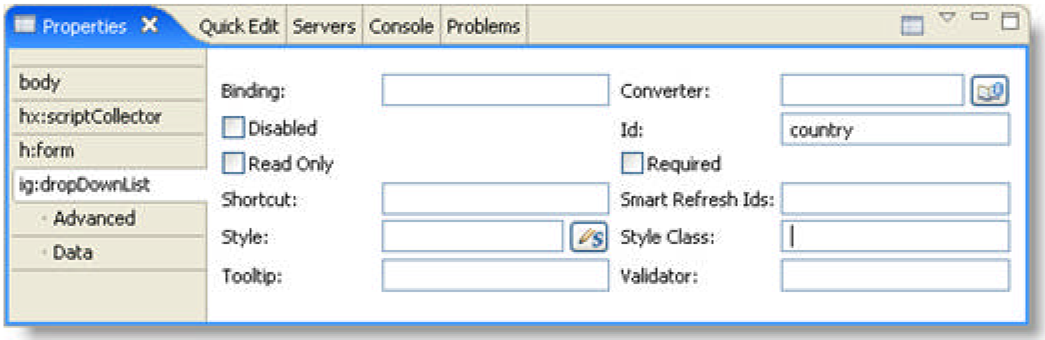
1. Open the <ig:dropDownList> tag and set the following attributes, as shown in the JSP code below:
   - id -- country
   - dataSource -- #{webinput_checkBoxList.countryList}

   In JSP:

```
<ig:dropDownList
    id = "country"
    dataSource ="#{webinput_checkBoxList.countryList}"/>
```

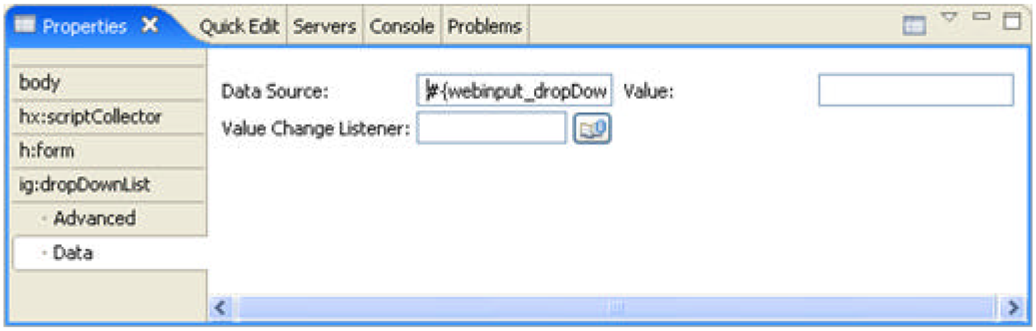
To create a DropdownList component using RAD:

1. From the palette, drag the DropDownList component to your page.
2. In the Properties window, select ig:dropDownList.
3. Set Id to country



Id: country

4. In the Properties tab, under ig:dropDownList, select Data.
5. Set Data Source to #{webinput_dropDown.countryList}



Data Source: #{webinput_dropDow

Value:

Value Change Listener:

4.8.1.5 InputCurrency

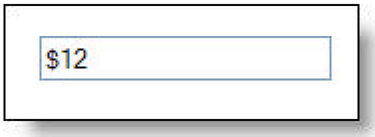
The InputCurrency component is used to display your end user's input in a currency format.

The InputCurrency component allows you to specify:

- the currency type.
- the maximum number of digits to be displayed after the decimal point.
- the minimum number of digits to be displayed after the decimal point.
- the maximum number of digits to be displayed before the decimal point.
- the minimum number of digits to be displayed before the decimal point.
- the maximum value allowed.
- the minimum value allowed.

This topic will demonstrate how to create the InputCurrency component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



$12

To create an InputCurrency component using code:

1. Open the <ig:inputCurrency> tag and set the following attributes, as shown in the JSP code below:
   - value -- 0
   - maximumFractionDigits -- 2
   - maximumIntegerDigits -- 4
   - maximumNumber -- 90
   - minimumFractionDigits -- 0
   - minimumIntegerDigits -- 2
   - minimumNumber -- 10
   - currency -- US

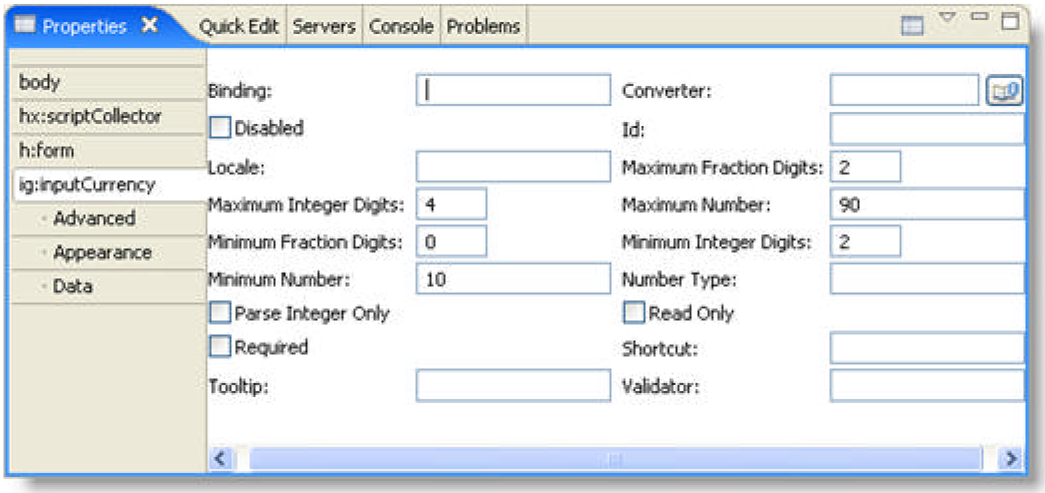
   In JSP:

```
<ig:inputCurrency value="0"
    maximumFractionDigits="2"
    maximumIntegerDigits="4"
    maximumNumber="90"
    minimumFractionDigits="0"
    minimumIntegerDigits="2"
    minimumNumber="10"
    currency="US"/>
```

To create an InputCurrency component using RAD:

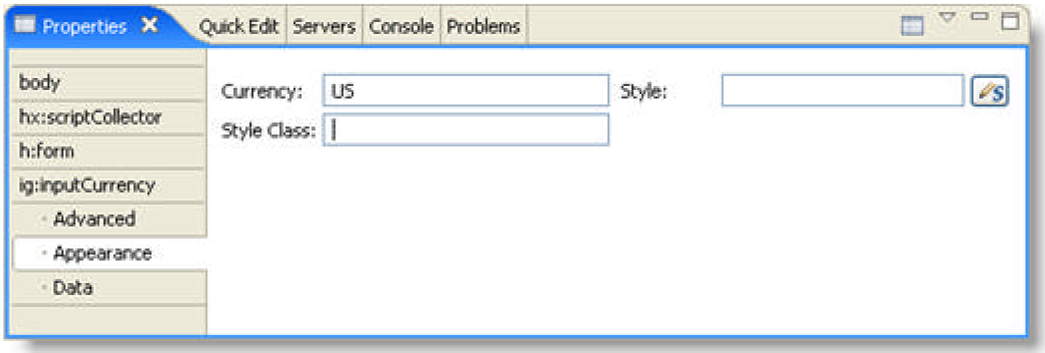
1. From the palette, drag the InputCurrency component to your page.

2. In the Properties window, select ig:inputCurrency.
3. Set the following properties:
   - Maximum Fraction Digits -- 2
   - Maximum Integer Digits -- 4
   - Maximum Number -- 90
   - Minimum Fraction Digits -- 0
   - Minimum Integer Digits -- 2
   - Minimum Number -- 10



Binding: |
Converter:
□ Disabled
Id:
Locale:
Maximum Fraction Digits: 2
Maximum Integer Digits: 4
Maximum Number: 90
Minimum Fraction Digits: 0
Minimum Integer Digits: 2
Minimum Number: 10
Number Type:
□ Parse Integer Only
□ Read Only
□ Required
Shortcut:
Tooltip:
Validator:

4. In the Properties tab, under ig:checkBoxList, select Appearance.
5. Set Currency to US.



Currency: US
Style:
Style Class: |

4.8.1.6 InputDecimal

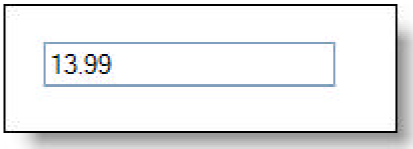
The InputDecimal component is used to display the end user's input in decimal format.

The InputDecimal component allows you to specify:

- the maximum number of digits to be displayed after the decimal point.
- the minimum number of digits to be displayed after the decimal point.
- the maximum number of digits to be displayed before the decimal point.
- the minimum number of digits to be displayed before the decimal point.
- the maximum value allowed.
- the minimum value allowed.

This topic will demonstrate how to use the InputDecimal component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create an InputDecimal component using code:

1. Open the <ig:inputDecimal> tag and set the following attributes, as shown in the JSP code below:
    - value -- 0
    - maximumFractionDigits -- 2
    - maximumIntegerDigits -- 4
    - maximumNumber -- 90
    - minimumFractionDigits -- 0
    - minimumIntegerDigits -- 2
    - minimumNumber -- 10

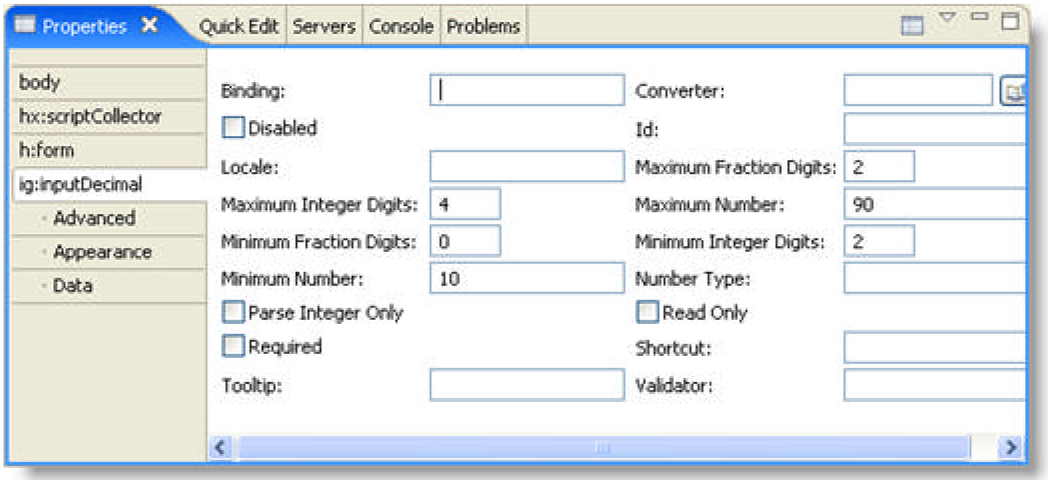
    In JSP:

```
<ig:inputDecimal value="0"
    maximumFractionDigits="2"
    maximumIntegerDigits="4"
    maximumNumber="90"
    minimumFractionDigits="0"
    minimumIntegerDigits="2"
    minimumNumber="10"/>
```

To create an InputDecimal component using RAD:

1. From the palette, drag the InputDecimal component to your page.
2. In the Properties window, select ig:inputDecimal.
3. Set the following properties:
    - Maximum Fraction Digits -- 2

- Maximum Integer Digits -- 4
- Maximum Number -- 90
- Minimum Fraction Digits -- 0
- Minimum Integer Digits -- 2
- Minimum Number -- 10



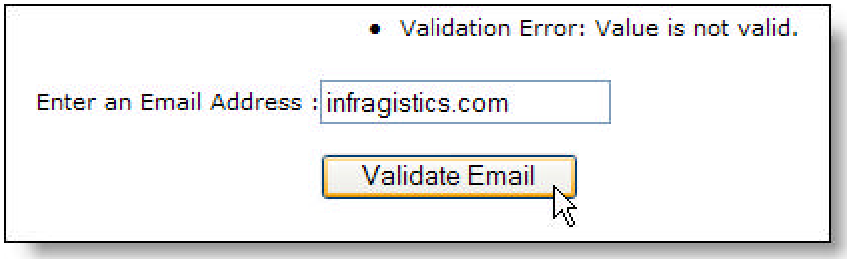
Binding:		Converter:	
□ Disabled		Id:	
Locale:		Maximum Fraction Digits:	2
Maximum Integer Digits:	4	Maximum Number:	90
Minimum Fraction Digits:	0	Minimum Integer Digits:	2
Minimum Number:	10	Number Type:	
□ Parse Integer Only		□ Read Only	
□ Required		Shortcut:	
Tooltip:		Validator:	

4.8.1.7 InputEmail

The InputEmail component is used to ensure that the end user has entered a valid e-mail address.

This topic will demonstrate how to use the InputEmail component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



- Validation Error: Value is not valid.

Enter an Email Address : infragistics.com

To create an InputEmail component using code:

1. Open the <ig:inputEmail> tag, as shown in the JSP code below:
   In JSP:

   ```
   <ig:inputEmail />
   ```

2. Create a commandButton, as shown in the JSP code below:
   In JSP:

   ```
   <h:commandButton value="Validate Email"
      title="decode inputEmail />
   ```

3. Save and run your application. Clicking on the button will automatically validate the input value.

To create an InputEmail component using RAD:

1. From the palette, drag the InputEmail component to your page.
2. From the palette, drag a command Button component to your page.
3. Save and run your application. Clicking on the button will automatically validate the input value.

4.8.1.8 InputPercent

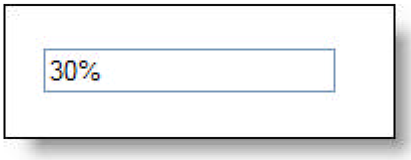
The InputPercent component is used to display the end user's input as a percentage.

The InputPercent component allows you to specify:

- the maximum number of digits to be displayed after the decimal point.
- the minimum number of digits to be displayed after the decimal point.
- the maximum number of digits to be displayed before the decimal point.
- the minimum number of digits to be displayed before the decimal point.
- the maximum value allowed.
- the minimum value allowed.

This topic will demonstrate how to create the InputPercent component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create an InputPercent component using code:

1. Open the <ig:inputPercent> tag and set the following attributes, as shown in the JSP code below:
   - value -- 0
   - maximumFractionDigits -- 2
   - maximumIntegerDigits -- 4
   - maximumNumber -- 90
   - minimumFractionDigits -- 0
   - minimumIntegerDigits -- 2
   - minimumNumber -- 10

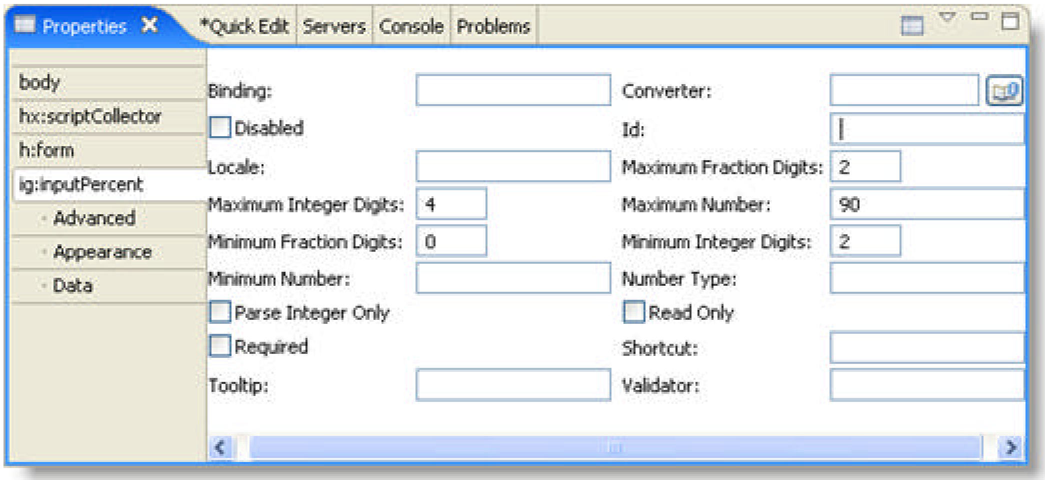
   In JSP:

```
<ig:inputPercent value="0"
    maximumFractionDigits="2"
    maximumIntegerDigits="4"
    maximumNumber="90"
    minimumFractionDigits="0"
    minimumIntegerDigits="2"
    minimumNumber="10"/>
```

To create an InputPercent component using RAD:

1. From the palette, drag the InputPercent component to your page.
2. In the Properties window, select ig:inputPercent.
3. Set the following properties:
   - Maximum Fraction Digits -- 2

- Maximum Integer Digits -- 4
- Maximum Number -- 90
- Minimum Fraction Digits -- 0
- Minimum Integer Digits -- 2
- Minimum Number -- 10



Binding:

□ Disabled

Locale:

Maximum Integer Digits: 4

Minimum Fraction Digits: 0

Minimum Number:

□ Parse Integer Only

□ Required

Tooltip:

Converter:

Id:

Maximum Fraction Digits: 2

Maximum Number: 90

Minimum Integer Digits: 2

Number Type:

□ Read Only

Shortcut:

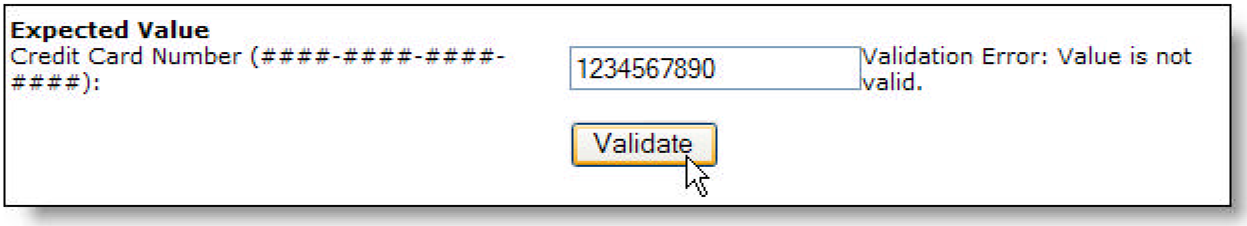
Validator:

4.8.1.9 InputRegularExpression

The InputRegularExpression component is used to create custom validation routines on the end user's input.

This topic will demonstrate how to use the InputRegularExpression component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Expected Value
Credit Card Number (####-####-####-####): 1234567890 Validation Error: Value is not valid.

To create an InputRegularExpression component using code:

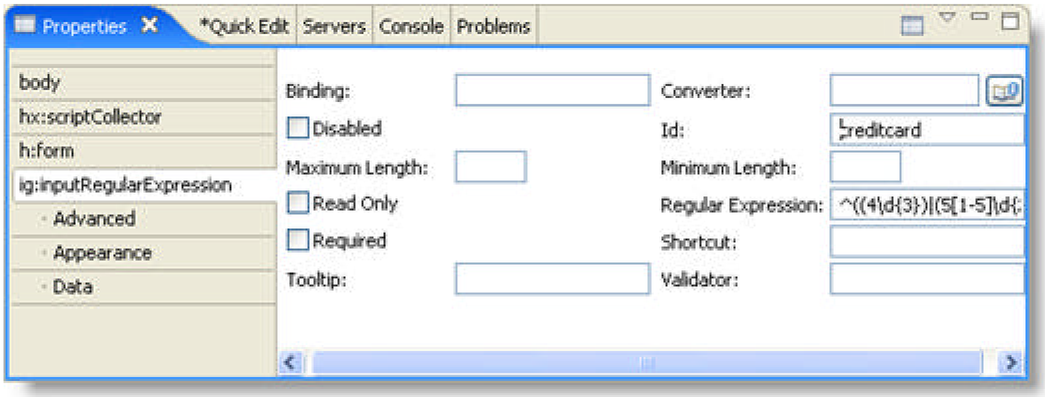
1. Open the <ig:inputRegularExpression> tag and set the following attributes, as shown in the JSP code below:
   - id -- creditcard
   - regularExpression -- ^((4\d{3})|(5[1-5]\d{2})|(6011))[-\s]?\d{4}[-\s]?\d{4}[-\s]?\d{4}|3[4,7]\d{13}$

   In JSP:

```
<ig:inputRegularExpression id="creditcard"
 regularExpression="^((4\d{3})|(5[1-5]\d{2})|(6011))[-\s]?\d{4}[-\s]?\d{4}[-\s]?\
/>
```

To create an InputRegularExpression component using RAD:

1. From the palette, drag the InputRegularExpression component to your page.
2. In the Properties window, select ig:inputRegularExpression.
3. Set the following properties:
   - id -- creditcard
   - regularExpression -- ^((4\d{3})|(5[1-5]\d{2})|(6011))[-\s]?\d{4}[-\s]?\d{4}[-\s]?\d{4}|3[4,7]\d{13}$



Binding:
□ Disabled
Maximum Length:
□ Read Only
□ Required
Tooltip:
Converter:
Id: creditcard
Minimum Length:
Regular Expression: ^((4\d{3})|(5[1-5]\d{
Shortcut:
Validator:

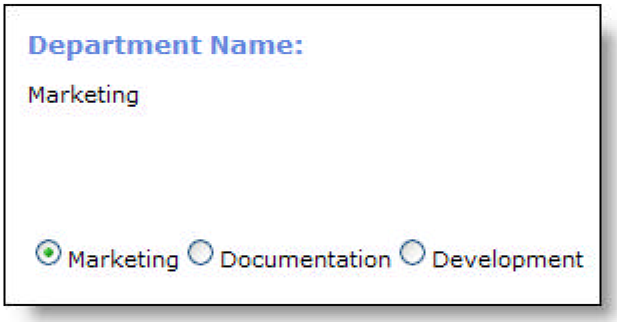
NetAdvantage for JSF

4.8.1.10 RadioButton

The RadioButton component is used for selection purposes in your application and can also be configured to update other components based on the values selected. Only one button per group can be selected.

This topic will demonstrate how to use the RadioButton component and display the value when the radio button is selected, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a RadioButton component using code:

1. Open the <ig:radioButton> tag and set the following attributes, as shown in the JSP code below:
   - binding -- #{webinput_radioButton.radioButton1}
   - groupName -- deptGroup
   - label -- Marketing
   - value -- radioButton1
   - smartRefreshIds -- outputSelectedRadioButton

   In JSP:

```
<ig:radioButton
    binding="#{webinput_radioButton.radioButton1}"
    groupName="deptGroup"
    label="Marketing"
    value="radioButton1"
    smartRefreshIds="outputSelectedRadioButton"/>
```

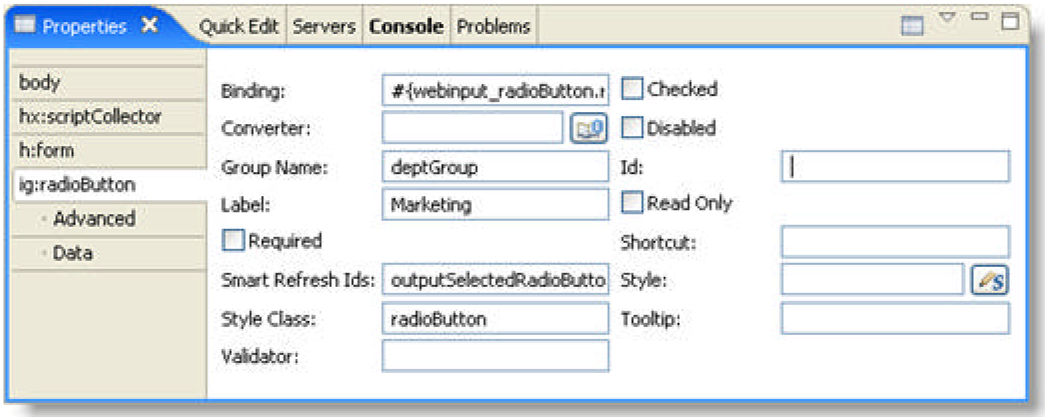
2. Repeat for each radio button.
3. Open the <h:outputText> tag and set the following attributes as shown in the JSP code below. This is where the values of the radio button will be displayed:
   - id -- outputSelectedRadioButton
   - value -- #{webinput_radioButton.selectedRadioButton}

   In JSP:

```
<h:outputText
    id="outputSelectedRadioButton"
    value="#{webinput_radioButton.selectedRadioButton}" />
```

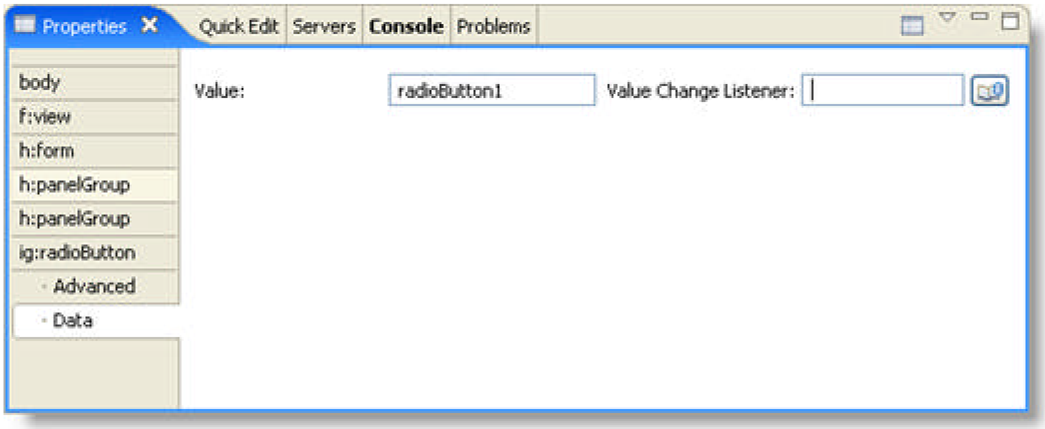
To create a RadioButton component using RAD:

1. From the palette, drag the RadioButton component to your page.
2. In the Properties window, select ig:radioButton.
3. Set the following properties:
   - Binding -- #{webinput_radioButton.radioButton1}
   - Group Name -- deptGroup
   - Label -- Marketing
   - Smart Refresh Ids -- outputSelectedRadioButton



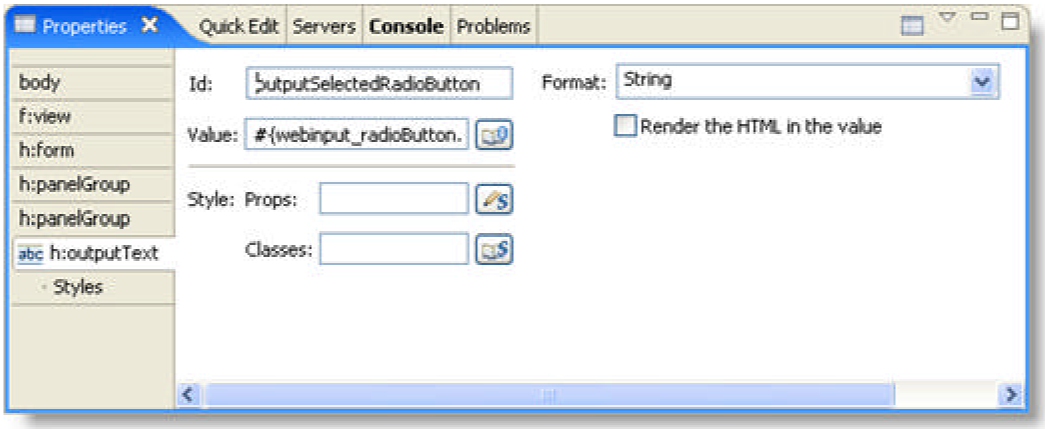
Binding:	#{webinput_radioButton.r	□ Checked	
Converter:		□ Disabled	
Group Name:	deptGroup	Id:	
Label:	Marketing	□ Read Only	
□ Required		Shortcut:	
Smart Refresh Ids:	outputSelectedRadioButto	Style:	
Style Class:	radioButton	Tooltip:	
Validator:			

4. In the Properties tab, under ig:radioButton, select Data.
5. Set Value to radioButton1.



Value: radioButton1 Value Change Listener:

6. Repeat for each radio button.
7. From the palette, under the Enhanced Faces Components, drag the Output component on to your page.
8. In the Properties window, select h:outputText.
9. Set the following properties:
   Id -- outputSelectedRadioButton
   Value -- #{webinput_radioButton.selectedRadioButton}
   Format -- String



Id: outputSelectedRadioButton

Value: #{webinput_radioButton.

Format: String

□ Render the HTML in the value

Style: Props:

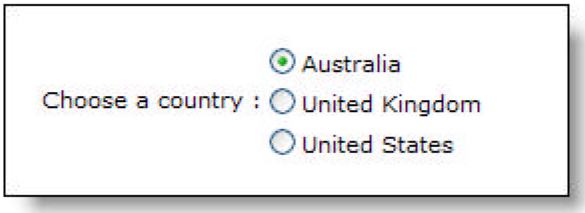
Classes:

4.8.1.11 RadioButtonList

The RadioButtonList component is used for selection purposes in your application and can also be configured to update other components based on the value selected.

This topic will demonstrate how to use the RadioButtonList component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



Choose a country : ● Australia ○ United Kingdom ○ United States

To create a RadioButtonList component using code:

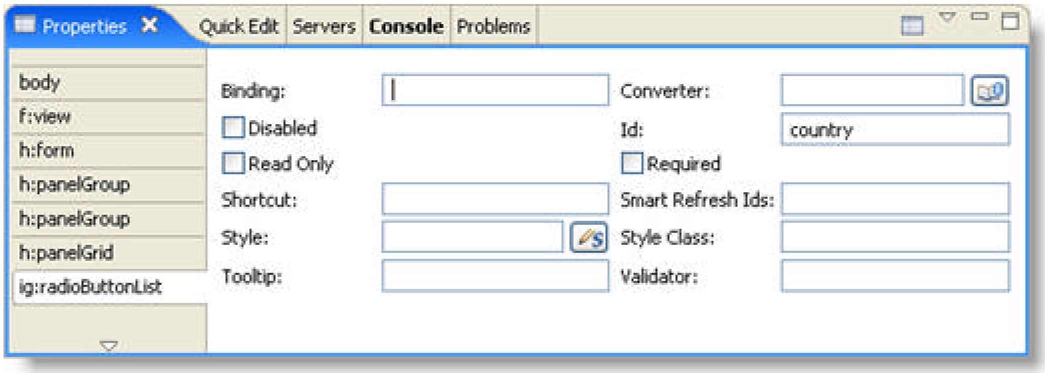
1. Open the <ig:radioButtonList> tag and set the following attributes, as shown in the JSP code below:
   - id -- country
   - dataSource -- #{webinput_radioButtonList.countryList}

   In JSP:

```
<ig:radioButtonList
    id = "country"
    dataSource ="#{webinput_radioButtonList.countryList}"/>
```

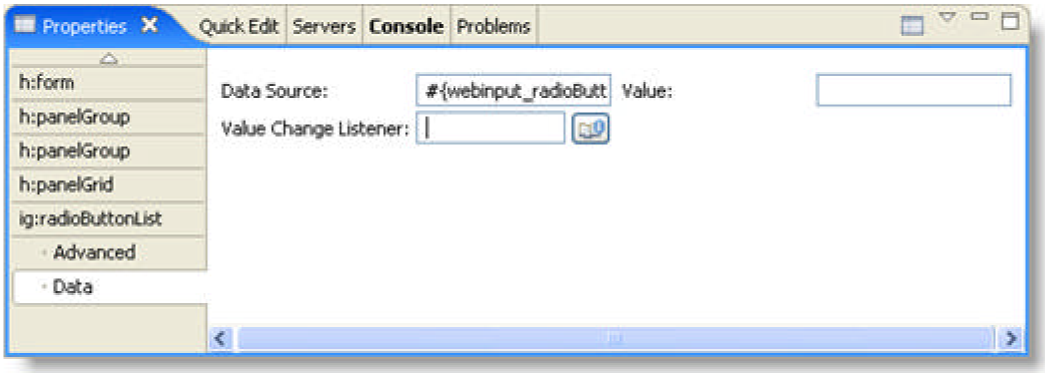
To create a RadioButtonList component using RAD:

1. From the palette, drag the RadioButtonList component to your page.
2. In the Properties window, select ig:radioButtonList.
3. Set Id to country



Binding:
□ Disabled
□ Read Only
Shortcut:
Style:
Tooltip:
Converter:
Id: country
□ Required
Smart Refresh Ids:
Style Class:
Validator:

4. In the Properties tab, under ig:radioButtonList, select Data.
5. Set Data Source to #{webinput_radioButtonList.countryList}



Data Source: #{webinput_radioButt

Value:

Value Change Listener:

4.8.2 WebInput Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.9 WebMenu

Click the links below to find information specific to the *WebMenu*™ components.

- **About the WebMenu Components (Section 4.9.1)** -- Provides information about the key functionalities of the WebMenu components.
- **Create a WebMenu Component (Section 4.9.2)** -- Contains a short, task-based topic that explains how to create a WebMenu component.
- **Expand or Collapse WebMenu with Mouse Events (Section 4.9.3)** -- This topic explains how to control how the WebMenu opens or closes.
- **WebMenu Accessibility Compliance (Section 4.9.4)** -- This topic outlines the way in which WebMenu meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebMenu control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.9.1 About the WebMenu Components

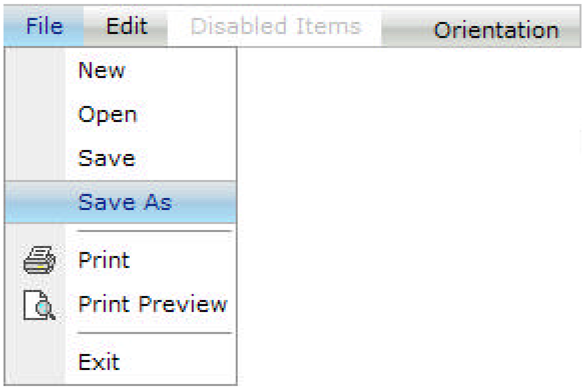
The following is a list of the *WebMenu*™ components:

- Menu
- MenuItem
- MenuItemCheckMark
- MenuItemSeparator

The WebMenu component allows you to customize the look and feel of the following UI elements:

- Appearance of the menu bar
- Default appearance of menu items nested inside a menu bar
- Hover appearance of menu items nested inside a menu bar
- Appearance of menus (other than the menu bar)
- Default appearance of menu items
- Hover appearance of menu items

Check marks or radio buttons can be displayed next to a menu item. They can also be grouped so that only one menu item is selected at a time.

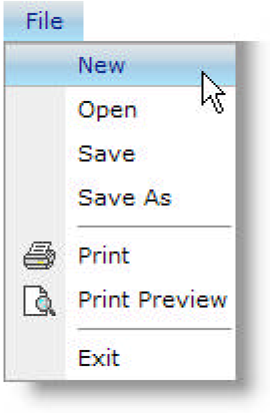


4.9.2 Create a WebMenu Component

The WebMenu component is a container component that displays MenuItems.

This topic will demonstrate how to create a WebMenu component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



To create a WebMenu component using code:

1. Open the <ig:menu> tag and set the id property to menu1, as shown in the JSP code below:
   In JSP:

   ```
   <ig:menu id="menu1">
   ```

2. Add a menu item and set the following properties:
   - value -- File
   - actionListener -- #{webmenu_overview.onItemClick}

   In JSP:

   ```
   <ig:menuItem value="File"
       actionListener="#{webmenu_overview.onItemClick}">
   ```

3. The following menuItems are nested within the top level menuItem File, so they will appear as a drop down menu. Add the nested menuItems and set the following properties:
   - menuItem 1
     - value -- New
     - actionListener -- #{webmenu_overview.onItemClick}
   - menuItem 2
     - value -- Open
     - actionListener -- #{webmenu_overview.onItemClick}
   - menuItem 3
     - value -- Save
     - actionListener -- #{webmenu_overview.onItemClick}

- menuItem 4
    - value -- Save As
    - actionListener -- #{webmenu_overview.onItemClick}
- Create a menuItemSeparator
- menuItem 5
    - value -- Print
    - iconUrl -- /resources/print.gif
    - hoverIconUrl -- /resources/printhover.gif
    - actionListener -- #{webmenu_overview.onItemClick}
- menuItem 6
    - value -- Print Preview
    - iconUrl -- /resources/printpreview.gif
    - hoverIconUrl -- /resources/printpreviewhover.gif
    - actionListener -- #{webmenu_overview.onItemClick}
- Create a menuItemSeparator
- menuItem 7
    - value -- Exit
    - actionListener -- #{webmenu_overview.onItemClick}

In JSP:

```
<ig:menuItem value="New"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItem value="Open"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItem value="Save"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItem value="Save As"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItemSeparator />
<ig:menuItem value="Print"
    iconUrl="/resources/print.gif"
    hoverIconUrl="/resources/printhover.gif"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItem value="Print Preview"
    iconUrl="/resources/printpreview.gif"
    hoverIconUrl="/resources/printpreviewhover.gif"
    actionListener="#{webmenu_overview.onItemClick}" />
<ig:menuItemSeparator />
<ig:menuItem value="Exit"
    actionListener="#{webmenu_overview.onItemClick}" />
```

4. Close the top level <ig:menuItem> tag.
   In JSP:

```
 </ig:menuItem>
```

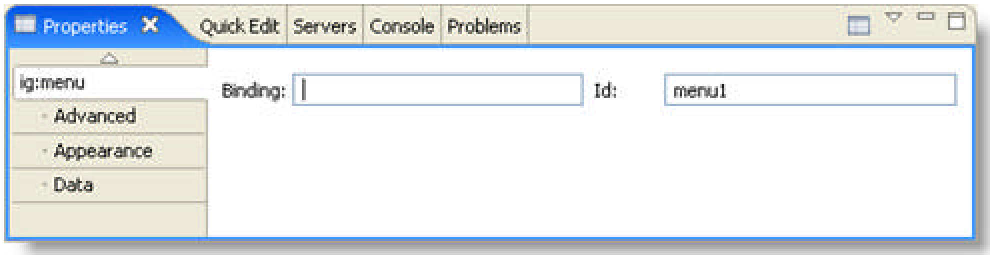
5. Repeat to add more menu items.

6. Close the <ig:menu> tag.
   In JSP:

```
</ig:menu>
```

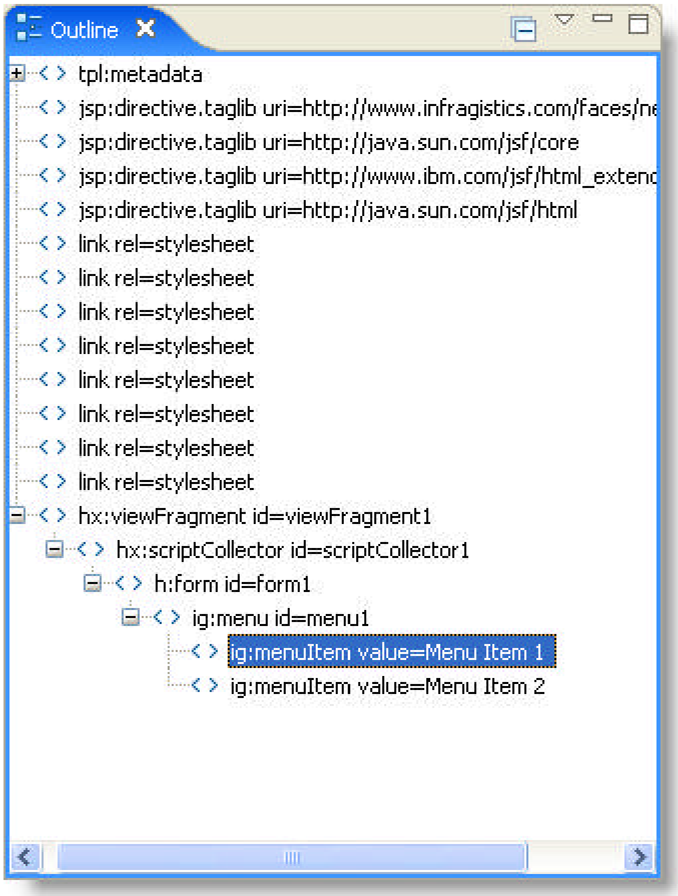
To create a WebMenu component using RAD:

1. From the palette, drag the Menu component to your page.
2. The default Menu is created with two menu items.
3. From the palette, drag the MenuItem component to your Menu to add more menu items.
4. In the Properties tab, set Id to menu1.



Binding:
Id: menu1

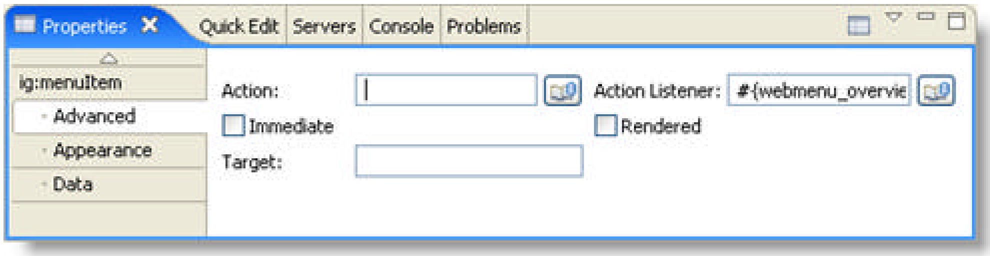
5. Expand your menu from the Outline tab, then select Menu Item 1.



```
tpl:metadata
jsp:directive.taglib uri=http://www.infragistics.com/faces/ne
jsp:directive.taglib uri=http://java.sun.com/jsf/core
jsp:directive.taglib uri=http://www.ibm.com/jsf/html_extend
jsp:directive.taglib uri=http://java.sun.com/jsf/html
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
hx:viewFragment id=viewFragment1
  hx:scriptCollector id=scriptCollector1
    h:form id=form1
      ig:menu id=menu1
        ig:menuItem value=Menu Item 1
        ig:menuItem value=Menu Item 2
```

6. In the Properties tab, under ig:menuItem, select Advanced.

7. Set Action Listener to #{webmenu_overview.onItemClick}.



Action:

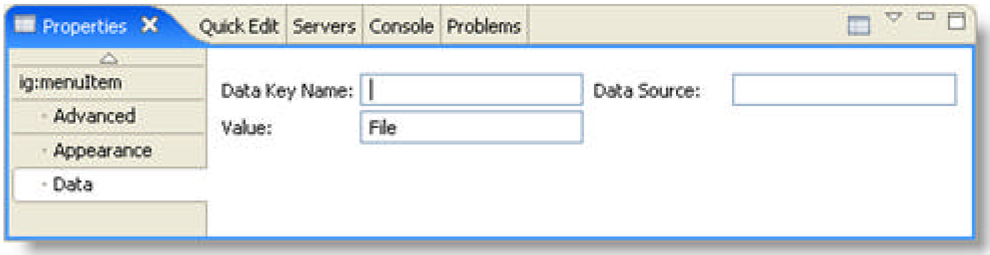
Action Listener: #{webmenu_overvie

□ Immediate

□ Rendered

Target:

8. In the Properties tab, under ig:menuItem, select Data.
9. Set Value to File.



Data Key Name:

Data Source:

Value: File

10. From the palette, drag the MenuItem component to the File menu item to add nested menu items.
11. Expand the File menu item from the Outline tab.
12. Set the following properties for each menu item:
    - menuItem 1
        - Value -- New
        - Action Listener -- #{webmenu_overview.onItemClick}
    - menuItem 2
        - Value -- Open
        - Action Listener -- #{webmenu_overview.onItemClick}
    - menuItem 3
        - Value -- Save
        - Action Listener -- #{webmenu_overview.onItemClick}
    - menuItem 4
        - Value -- Save As
        - Action Listener -- #{webmenu_overview.onItemClick}
    - Create a menuItemSeparator
    - menuItem 5
        - Value -- Print
        - Icon Url -- /resources/print.gif
        - Hover Icon Url -- /resources/printhover.gif
        - Action Listener -- #{webmenu_overview.onItemClick}
    - menuItem 6

    - Value -- Print Preview
    - Icon Url -- /resources/printpreview.gif
    - Hover Icon Url -- /resources/printpreviewhover.gif
    - Action Listener -- #{webmenu_overview.onItemClick}
  - Create a menuItemSeparator
  - menuItem 7
    - Value -- Exit
    - Action Listener -- #{webmenu_overview.onItemClick}

13. Repeat to add more menu items.

4.9.3 Expand or Collapse WebMenu with Mouse Events

The expandOn and collapseOn attributes of the <ig:menu> and <ig:menuItem> tag specifies how to control when the menu items opens or closes through mouse events.

Currently, the expandOn attribute accepts a mouse over event or a mouse left click event as values.

The collapseOn attribute accepts a mouse hover out event or a mouse click out event as values.

You can set these attributes at the parent level, so that all the children menu items will inherit the setting or you can set it for each individual menu item.

The following code demonstrates how to open a menu item by using the mouse left click event.

In JSP:

```
<ig:menuItem expandOn="leftMouseClick">
   <ig:menuItem> ...
   <ig:menuItem> ...
</ig:menuItem>
```

4.9.4 WebMenu Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.10 WebTab

Click the links below to find information specific to the *WebTab*™ components.

- **About the WebTab Components (Section 4.10.1)** -- Provides information about the key functionalities of the WebTab components.
- **Create a WebTab Component (Section 4.10.2)** -- Contains a short, task-based topic that explains how to create a WebTab component.
- **WebTab Accessibility Compliance (Section 4.10.3)** -- This topic outlines the way in which WebTab meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebTab control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.10.1 About the WebTab Components

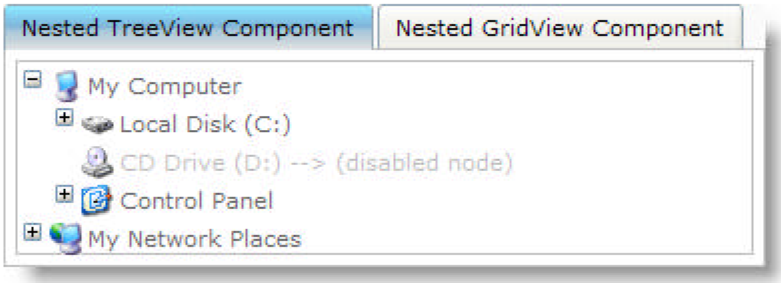
The following is a list of the *WebTab*™ components:

- TabView
- TabItem

The WebTab component allows you to customize the look and feel of the following UI elements:

- Default appearance of a TabView
- Default appearance of a TabItem
- Hover appearance of TabItems
- Selected appearance of TabItems
- Disabled TabItems

These components support a nested TabView, providing the ability to display tabs within tabs for the more complex configurations that your application may require.


Nested TreeView Component
Nested GridView Component
My Computer
Local Disk (C:)
CD Drive (D:) --> (disabled node)
Control Panel
My Network Places

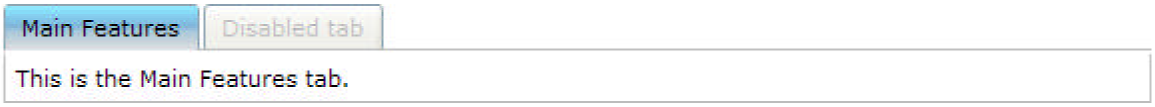
4.10.2 Create a WebTab Component

The WebTab component contains a TabView component and a TabItem component. The TabView component is a container component that displays TabItems.

WebTab automatically and transparently sends AJAX requests to fetch TabItems On-Demand.

This topic will demonstrate how to use the WebTab component, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



This is the Main Features tab.

To create a WebTab component using code:

1. Open the <ig:tabView> tag, as shown in the JSP code below:
   In JSP:

```
<ig:tabView>
```

2. Add a tab item, set the value property to Main Features
   In JSP:

```
<ig:tabItem value="Main Features">
```

3. Enter the content to be displayed in that tab. Close the tab item.
   In JSP:

```
  <f:verbatim escape="false">
    This is the Main Features Tab
  </f:verbatim>
</ig:tabItem>
```

4. Create another tab item, set the following properties:
   - value -- Disabled Tab
   - disabled -- true

   In JSP:

```
 <ig:tabItem value="Disabled Tab" disabled="true">
```

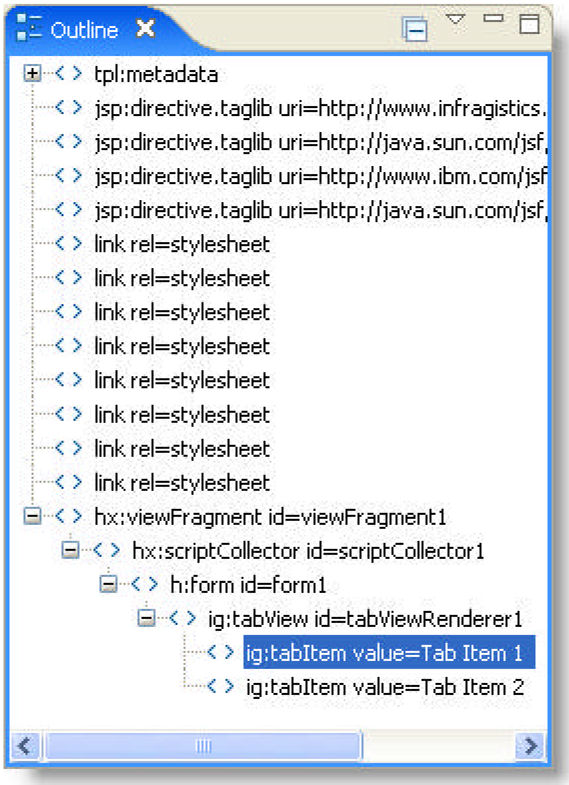
5. Close the <ig:tabView> tag.
   In JSP:

```
 </ig:tabView>
```

To create a WebTab component using RAD:

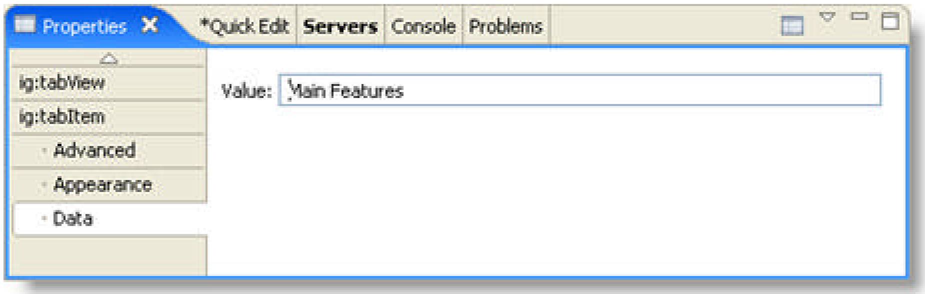
1. From the palette, drag the TabView component to your page.
2. The default TabView is created with two tab items. From the palette, drag the TabItem component to your TabView to add more tab items.

3. Expand your TabView from the Outline tab, then select Tab Item 1.



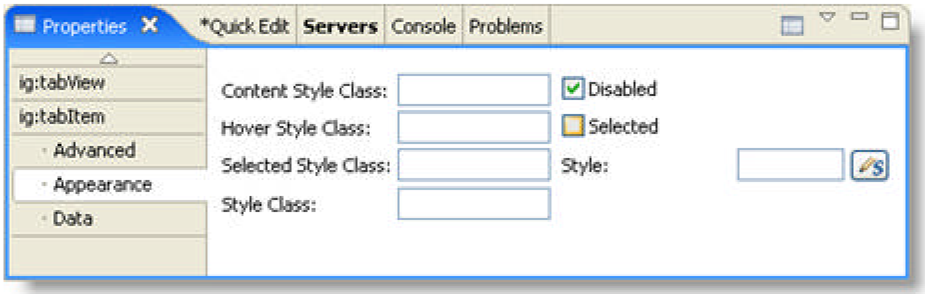
```
tpl:metadata
jsp:directive.taglib uri=http://www.infragistics.
jsp:directive.taglib uri=http://java.sun.com/jsf,
jsp:directive.taglib uri=http://www.ibm.com/jsf
jsp:directive.taglib uri=http://java.sun.com/jsf,
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
link rel=stylesheet
hx:viewFragment id=viewFragment1
  hx:scriptCollector id=scriptCollector1
    h:form id=form1
      ig:tabView id=tabViewRenderer1
        ig:tabItem value=Tab Item 1
        ig:tabItem value=Tab Item 2
```

4. In the Properties tab, under ig:tabItem, select Data.
5. Set Value to Main Features.



Value: Main Features

6. From the Outline Tab, select Tab Item 2.
7. In the Properties tab, under ig:tabItem, select Appearance.
8. Select the Disabled check box.



Content Style Class:

Hover Style Class:

Selected Style Class:

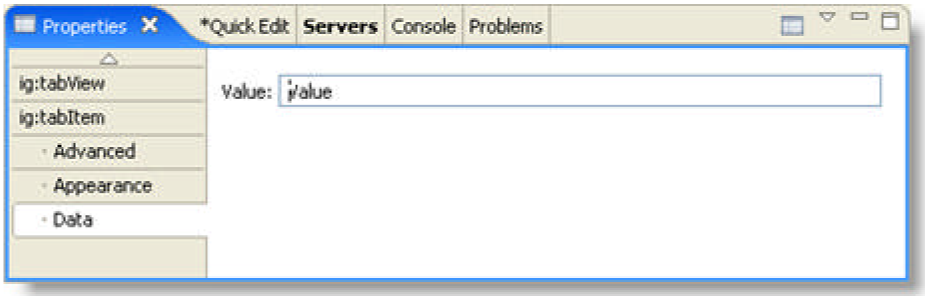
Style Class:

☑ Disabled

□ Selected

Style:

9. In the Properties tab, under ig:tabItem, select Data.
10. Set Value to Disabled.



Value: value

4.10.3 WebTab Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

4.11 WebTree

Click the links below to find information specific to the *WebTree*™ components.

- **About the WebTree Components (Section 4.11.1)** -- Provides information about the key functionalities of the WebTree components.
- **WebTree Accessibility Compliance (Section 4.11.2)** -- This topic outlines the way in which WebTree meets the requirements of the Section 508 standards. This accessibility compliance allows your end users, specifically those with disabilities, to be able to access and use your Web application built using the WebTree control.
- **API Reference Guide (on-line documentation)** -- The Application Programming Interface (API) Reference Guide is the foundation for all our NetAdvantage for JSF components and offers a definitive listing of all the NetAdvantage for JSF classes and interfaces.

4.11.1 About the WebTree Components

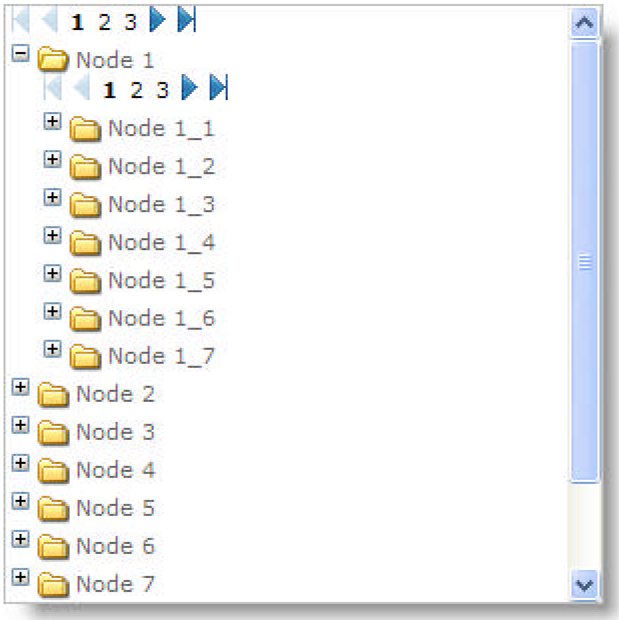
The following is a list of the *WebTree*™ components:

- TreeView
- TreeNode

These components allow you to perform various tasks, including:

- add items to the TreeView
- add nested items to the TreeView
- customize the TreeView's look and feel
- change a TreeView item's icon or label

The WebTree components provide AJAX-enabled automatic paging. They can automatically display and handle the First, Previous, Next, and Last buttons. You simply need to set a property to specify how many nodes you want your end users to page though using a single mouse-click. The pager automatically displays if the tree has more nodes that that which is specified in the page size. The Smart Refresh™ technology on which the automatic paging is based eliminates postbacks and reduces network traffic.



Note: When collapsing a node, a server call is not issued as the data is already on the client-side, therefore the CollapseListener on the server will get queued and will get called but not immediately. Also, any subsequent expands after the initial expand will not call the server-side's listener method immediately.

4.11.2 WebTree Accessibility Compliance

All of our JSF components comply with section 508, subpart 1194.21 of the Rehabilitation Act of 1973. Below is a table containing the rules of subpart 1194.21 and how we comply with each rule.

Rules	How We Comply with the Rules
(a)	Infragistics' components include keyboard support for tabbing into controls and navigating within them. All NetAdvantage for JSF components that accept focus provide keyboard support.
(b)	Infragistics NetAdvantage for JSF enables individuals to customize their desktop elements, including resizing and rearranging controls and choosing color, size and format options.
(c)	NetAdvantage for JSF provides a well-defined on-screen indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions. The components provide hover styles and selection styles that highlight the position of the focus on the form or page. Focus rectangles are also used along with the system caret to indicate focus location within the elements.
(d)	Infragistics NetAdvantage for JSF provides sufficient information about user interface elements and program elements represented by images are available in text in virtually all cases.
(g)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(i)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(j)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.
(k)	Infragistics NetAdvantage for JSF provides functionality that conforms to this criteria.

5 Designing the Look and Feel

To enable a consistent end-user experience, the NetAdvantage for JSF components have been designed to share a number of style-related attributes and behaviors. Click the links below for information on the attributes common to almost all the NetAdvantage for JSF components.

- **Enabling and Disabling Components (Section 5.1)** – Explains how you can visually disable a WebInput component.
- **Setting Component ToolTips (Section 5.2)** – Shows you how to set a ToolTip for a WebInput component.
- **Themes, Styles and Style Classes (Section 5.3)** - Provides information on how to apply themes and use the style and styleClass attributes.
- **Apply a Style Class to a JSF Component (Section 5.4)** -- Provides a short, task-based topic that explain how to apply a style class to a JSF component.
- **Apply a Style to a JSF Component (Section 5.5)** -- Provides a short, task-based topic that explain how to apply a style to a JSF component.
- **Apply a Theme to a JSF Component (Section 5.6)** -- Provides a short, task-based topic that explain how to apply a theme to a JSF component.
- **Override Themes (Section 5.7)** -- Provides a short, task-based topic that explain how to override themes.

5.1 Enabling and Disabling Components

All the NetAdvantage for JSF components that accept user input (i.e., **the WebInput components (Section 4.8.1)**) also support global enabling and disabling. Unless specified otherwise, components are enabled.

In general, end users cannot interact with a disabled component in any way, and the component appears visually disabled.

Note: The detailed appearance of the disabled component depends on the component's style.

The sample code below demonstrates how to disable a component.

In Java:

```
DateChooser dateChooser = new DataChooser();
boolean disabled=dateChooser.isDisabled();
dateChooser.setDisabled(false);
```

In JSP:

```
<ig:DateChooser disabled="false"
  <%-- other attributes as required --%>
</ig:DateChooser>
```

Note: The Enabled status can be bound in the standard way using the "#{…}" notation.

5.2 Setting Component ToolTips

All the NetAdvantage for JSF components that accept user input (i.e., the *WebInput™* components) also support a ToolTip (i.e., a small yellow box containing descriptive text that appears when the end user hovers the mouse over the component).

The sample code below demonstrates how to set a ToolTip for a component.

In Java:

```
DateChooser dateChooser = new DateChooser();
String ToolTip=dateChooser.getToolTip();
```

In JSP:

```
<ig:DateChooser ToolTip="blue"
  <%-- other attributes as required --%>
</igDateChooser>
```

Note: The ToolTip value can be bound in the standard way using the “#{…}” notation.

5.3 Themes, Styles and Style Classes

You can use the available themes, as well as the style attribute and the styleClass attribute to define the look and feel of your application.

Themes

The NetAdvantage for JSF components support a set of predefined application-wide themes that correspond to styles commonly used in Web applications. These themes help to maintain a consistent look and feel throughout your application. For more information, see **Apply a Theme to a JSF component (Section 5.6)**.

Styles

You can use the style attribute to define the style of the NetAdvantage for JSF components. You specify the style within the style attribute, and the style applies only to the specific component. When you specify the style attribute for a component, you override the application-wide theme. For example, if the style attribute is defined within a <column> tag, then only that column will have that style applied to it. For more information, see **Apply a Style to a JSF component (Section 5.5)**.

Style Classes

You can use the styleClass attribute to define the style of the component. Create the style specifications within the igf_grid.css file of your specified theme. Then, using the styleClass attribute, you can implement the style. When you specify the styleClass attribute for a component, you override the application-wide theme. If you want to implement a particular style frequently, you would use the styleClass attribute instead of the style attribute. For more information, see **Apply a Style Class to a JSF component (Section 5.4)**.

Related Topics

Apply a Theme to a JSF component (Section 5.6)

Apply a Style to a JSF component (Section 5.5)

Apply a Style Class to a JSF component (Section 5.4)

5.4 Apply a Style Class to a JSF Component

The styleClass attribute defines the look and feel of your JSF components by allowing you to change the appearance of the component such as font, color and position. Applying the styleClass attribute to your component is different from overriding a theme as the style is only reflected when you call that class.

The following example, using the WebGrid component, creates a class that changes the color to red, the class is called from the column component.

To apply styles to WebGrid:

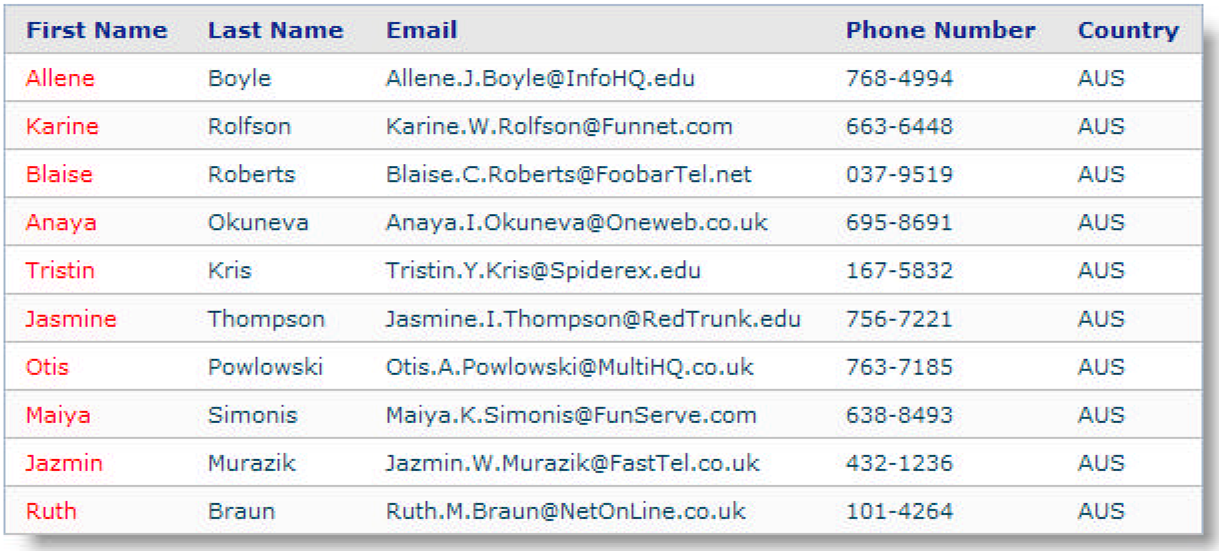
1. Add the following style class to the igf_grid.css file of your specified theme:
   In CSS:

```
.igRedColumn {
      border-bottom: 1px solid #C1C1C1;
      padding: 5px 10px 5px 10px;
      text-align: left;
      color : red;
      white-space: nowrap;
}
```

2. Add the styleClass attribute to the <h:outputText> tag:
   In JSP:

```
<h:outputText styleClass="igRedColumn" value="#{DATA_ROW.firstName}" />
```

3. The column font has changed to red.



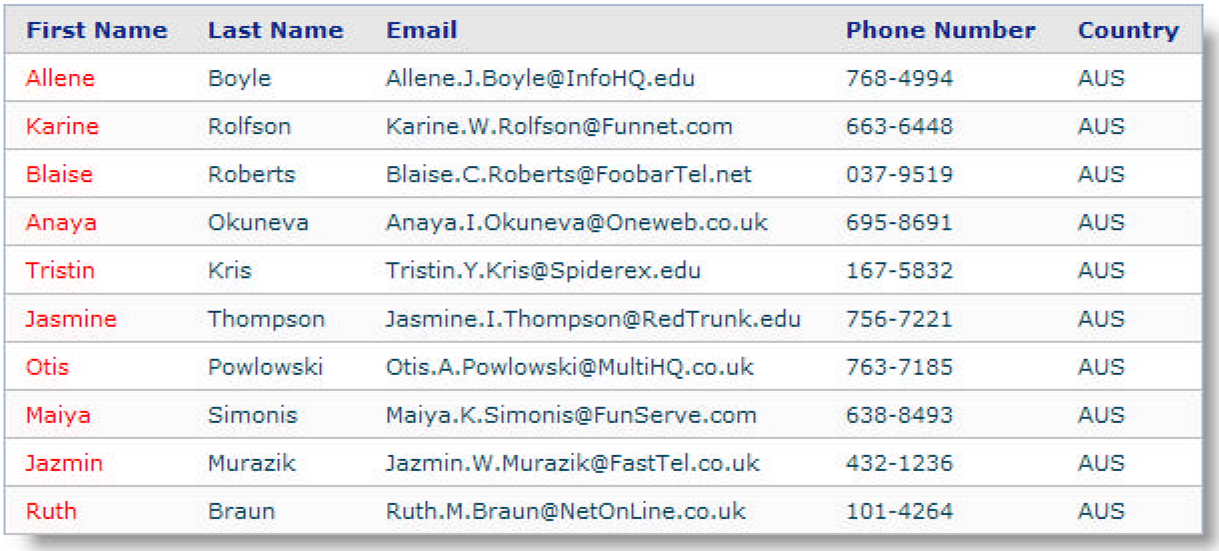
First Name	Last Name	Email	Phone Number	Country
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994	AUS
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448	AUS
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519	AUS
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691	AUS
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832	AUS
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221	AUS
Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185	AUS
Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493	AUS
Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236	AUS
Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264	AUS

5.5 Apply a Style to a JSF Component

The style attribute defines the look and feel of your JSF components by allowing you to change the appearance of the grid such as font, color and position of the grid components. You can define the style attribute of a component directly in your JSP page and it will apply only to that component in that page. Applying the style attribute to your component is different from overriding a theme as the change is not reflected application-wide.

This topic will demonstrate how to change the font color of the first column of a grid to red, as illustrated in the screen shot below:

- **using code**
- **at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0**



First Name	Last Name	Email	Phone Number	Country
Allene	Boyle	Allene.J.Boyle@InfoHQ.edu	768-4994	AUS
Karine	Rolfson	Karine.W.Rolfson@Funnet.com	663-6448	AUS
Blaise	Roberts	Blaise.C.Roberts@FoobarTel.net	037-9519	AUS
Anaya	Okuneva	Anaya.I.Okuneva@Oneweb.co.uk	695-8691	AUS
Tristin	Kris	Tristin.Y.Kris@Spiderex.edu	167-5832	AUS
Jasmine	Thompson	Jasmine.I.Thompson@RedTrunk.edu	756-7221	AUS
Otis	Powlowski	Otis.A.Powlowski@MultiHQ.co.uk	763-7185	AUS
Maiya	Simonis	Maiya.K.Simonis@FunServe.com	638-8493	AUS
Jazmin	Murazik	Jazmin.W.Murazik@FastTel.co.uk	432-1236	AUS
Ruth	Braun	Ruth.M.Braun@NetOnLine.co.uk	101-4264	AUS

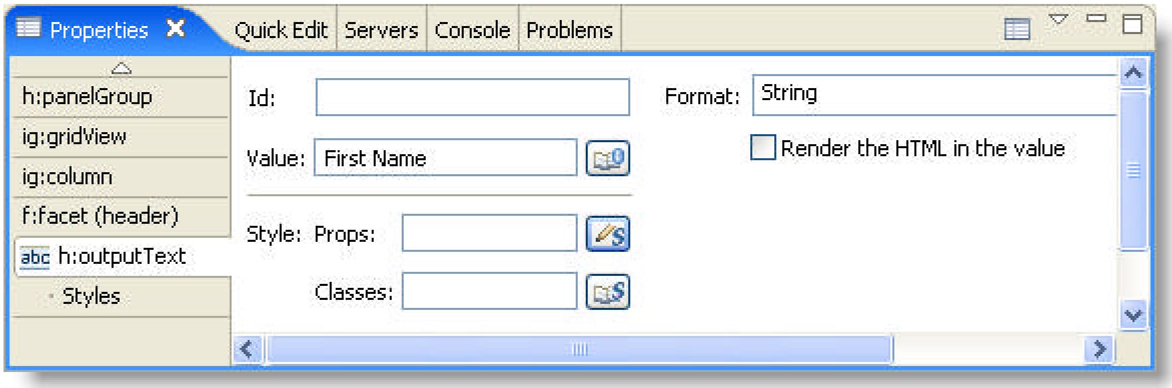
To apply the style using code:

In JSP:

```
<h:outputText style="color: red" value="#{DATA_ROW.firstName}" />
```

To apply the style at design time using RAD:

1. In Design view, click the column on the grid component on your page.



Id:

Value: First Name

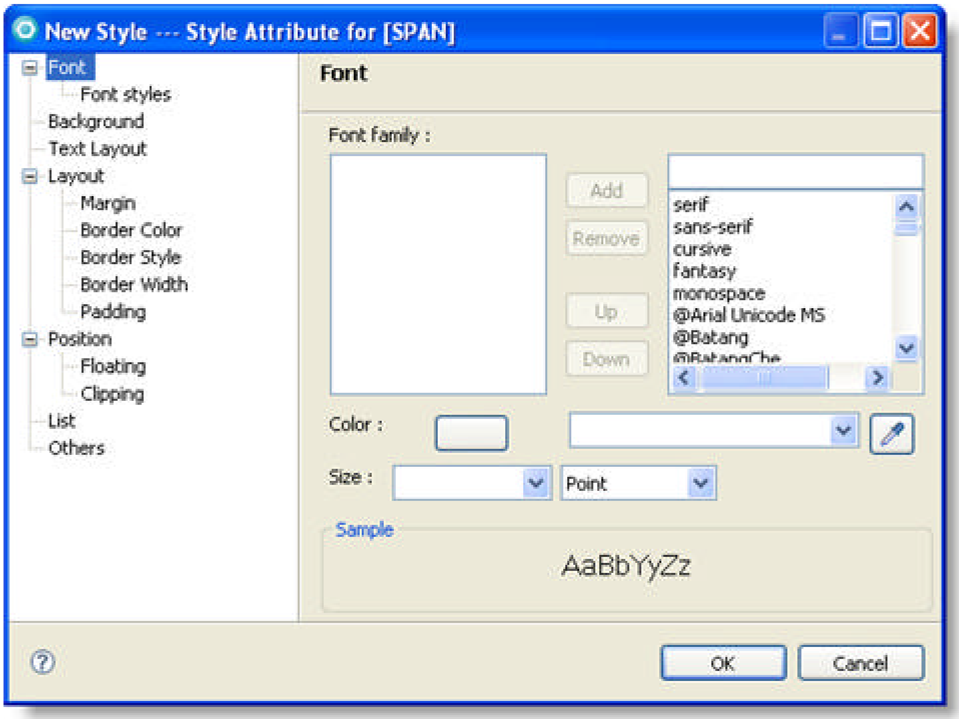
Format: String

☐ Render the HTML in the value

Style: Props:

Classes:

2. In the Properties window, next to the Style: Props box, click the edit button.



Font family :

Color :

Size : Point

3. Click the button next to the Color field, and select Red from the color palette that appears.

4. Click OK.

5.6 Apply a Theme to a JSF Component

Applying a theme to your application is a great vehicle for maintaining a consistent look and feel throughout your application. For example, specifying the Appletini theme applies a light green color to the JSF components.

You specify the theme in the web.xml file, located in the WEB-INF folder.

> **Note:** The themes are case-sensitive. By default, the Blue theme is applied to NetAdvantage for JSF components.

To apply the Appletini theme to a JSF component:

1. Open the web.xml file.
2. Add a new <context-param> element under the root element <web-app>, as shown in the XML excerpt below.
   In XML:

   ```
   <context-param>
      <param-name>com.infragistics.faces.THEME</param-name>
      <param-value>Appletini</param-value>
   </context-param>
   ```

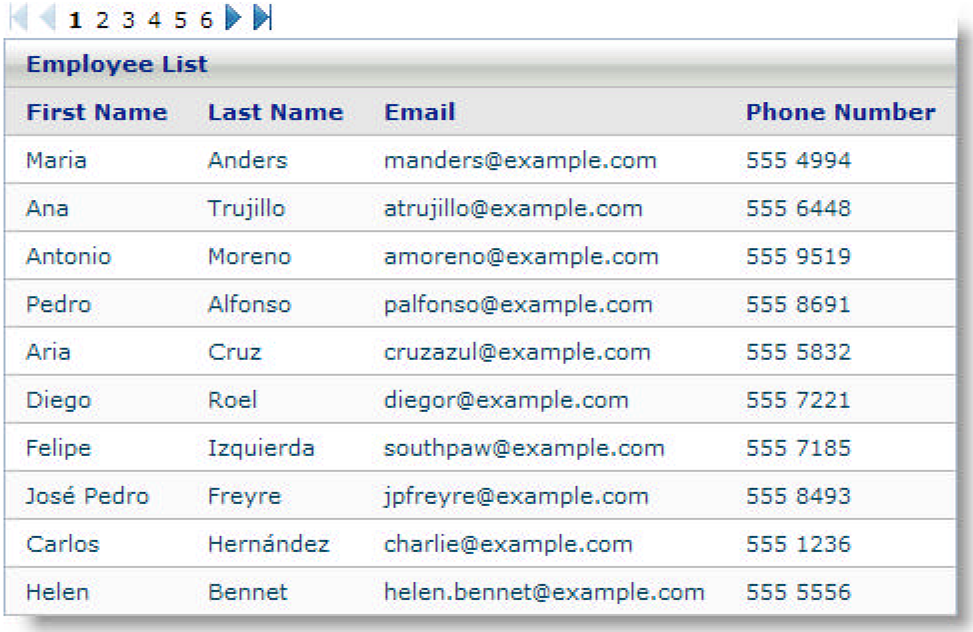
3. To change the theme modify the text of the <param-value> element.

5.7 Override Themes

If you have applied the available NetAdvantage for JSF themes to your application, but want to change a specific part of the theme, you need to override the theme.

For example if you applied the Blue theme to your grid but wanted the grid to have a 'grid-like' appearance with the columns separated by a line, you can simply override the settings in the igf_grid.css file of the Blue theme.

The following screen shot shows the grid before the Blue theme is overridden.



1 2 3 4 5 6

Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

To override the column settings of a theme:

1. Add a border to the grid column header class.
   In CSS:

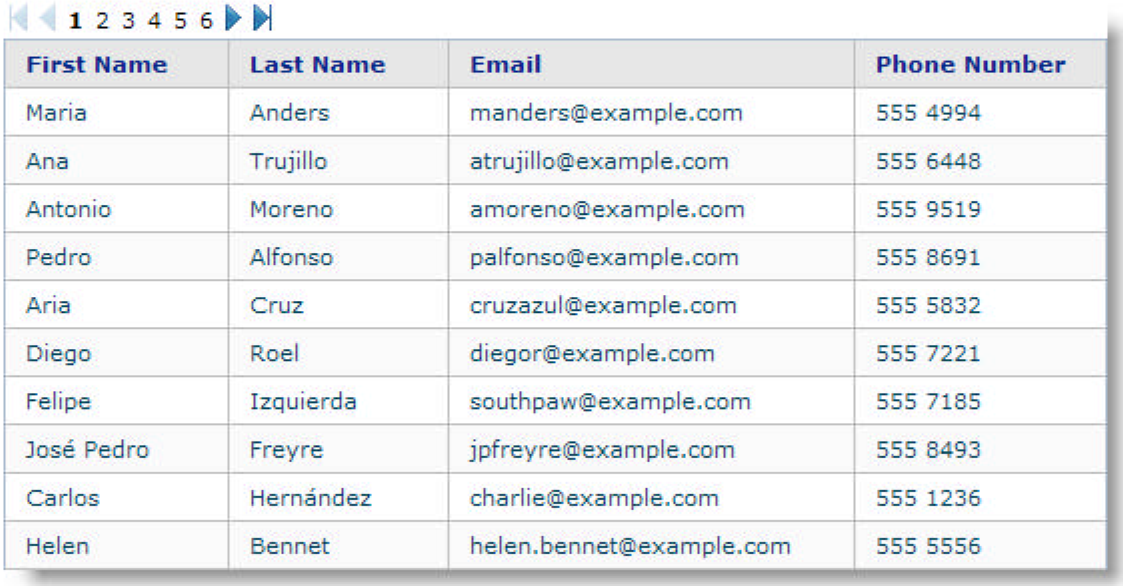
```
.igGridColumnHeader {
background-color: #e6e6e6;
      border-bottom: 1px solid #C1C1C1;
      border-right: 1px solid #C1C1C1;
      padding: 5px 10px 5px 10px;
      font-weight: bold;
      text-align: left;
      color : #052684;
      white-space: nowrap;
}
```

2. Add a border to the grid column class.
   In CSS:

```
.igGridColumn {
        border-bottom: 1px solid silver;
        border-right: 1px solid #C1C1C1;
        padding: 5px 10px 5px 10px;
```

```
            white-space: nowrap;
    }
```

3. Run the application. Your grid should look similar to the grid below, with the columns divided, giving a more grid-like appearance.



First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

6 Smart Refresh Technology

The NetAdvantage for JSF components come with a very robust and powerful Asynchronous JavaScript and XML (AJAX) framework. AJAX requests are performed behind the scenes and they eliminate full Web page refreshes.We refer to this as our Smart-Refresh® technology. This enables you to create better, faster, and more user-friendly web applications. Using the Smart-Refresh technology eliminates the need for you to write JavaScript code to perform AJAX requests; instead all you need to do is simply add the smartRefreshIds attribute to your NetAdvantage for JSF component.

For situations where you are using core JSF components or HTML tags, you can use the smartSubmit() method.

The smartRefreshIds attribute accepts a comma-separated list of all the IDs of the components you want updated. These IDs can belong to NetAdvantage® for JSF components, core JSF components such as or regular HTML tags.

The following topics describe different usages of the Smart-Refresh technology:

- **Use Smart-Refresh technology with NetAdvantage for JSF components (on-line documentation)**
- **Use Smart-Refresh technology with core JSF components (on-line documentation)**
- **Use Smart-Refresh technology without NetAdvantage for JSF components (on-line documentation)**

The following topic is an in-depth tutorial guiding you through creating a web application that implements the Smart-Refresh technology to perform AJAX requests.

- **WebInput Tutorial (Section 7.2.5)**

6.1 Using Smart-Refresh Technology with Core JSF Components

You can also perform AJAX requests using the smart-refresh technology from any core JSF component or even from an ordinary HTML tag. In situations like this, you can simply call the smartSubmit() method.

> **Note:** In order to avail ourselves to the smartSubmit() method, which is part of the Smart-Refresh technology used to perform AJAX requests, a NetAdvantage for JSF component must be present on your web page.

The smartSubmit() method accepts the following parameters:

- eventSource – The component that is calling the AJAX request. Cannot be null.
- eventName – A name associated with the event. Can be null.
- eventArguments – A set of arguments to be passed to the server. Can be null.
- smartRefreshComponents –The list of components to be updated. Cannot be null.
- jsCallback – The JavaScript method that is called when the AJAX request completes. Can be null.

The following code demonstrates how to call the smartSubmit() method from an HTML tag.

In JSP:

```
<input
   Id=”myButton”
   type=”button”
   value =”Update Time”
   onclick=”ig.smartSubmit(‘AJAXFromHTMLForm’, null, null, ‘AJAXFromHTMLForm:showTim
```

Related Topics

Using Smart-Refresh Technology with NetAdvantage for JSF Components (Section 6.2)

Using Smart-Refresh Technology without NetAdvantage for JSF Components (Section 6.3)

6.2 Using Smart-Refresh Technology with NetAdvantage for JSF Components

Using the Smart-Refresh technology with NetAdvantage for JSF components ensures that only the components you specify will update, not the entire page.

Consider a page that contains two dropdown menus. One has an ID of States and it is populated with all of the states of a country. The other dropdown menu has an ID of Cities and its content is based upon whatever is currently selected from the States dropdown menu. Whenever the end user selects a State from the States dropdown, the page will normally perform a full page postback and at that time, when the page is reloaded, the Cities dropdown will be populated with the Cities that belong to the selected State.

If you set the smartRefreshIds attribute of the States dropdown to the value “Cities”, the Cities dropdown will update using the Smart-Refresh technology.

The following code demonstrates how to implement this scenario.

In JSP:

```
<ig:dropDownList
   id="States"
   dataSource="#{dropdown.statesData}"
   binding="#{dropdown.statesList}"
   smartRefreshIds="Cities"
/>
<ig:dropDownList
   id="Cities"
   dataSource="#{dropdown.citiesData}"
   binding="#{dropdown.citiesList}"
/>
```

Related Topics

Using Smart-Refresh Technology with Core JSF Components (Section 6.1)

Using Smart-Refresh Technology without NetAdvantage for JSF Components (Section 6.3)

6.3 Using Smart-Refresh Technology without NetAdvantage for JSF Components

There may be certain situations where you want to use our Smart-Refresh technology to perform AJAX requests on a page where you do not have NetAdvantage for JSF components present. In order to achieve this you simply add the following JavaScript code snippet below the tag on your page. This references the necessary JavaScript files to complete seamless AJAX requests.

In JSP:

```
<script
   type="text/JavaScript"
   src="/INSTALL_DIR/resources/infragistics/scripts/igf_bootstrap.js"
>
</script>
<script type="text/JavaScript">
<!--
   igBootstrap("/INSTALL_DIR/resources/infragistics/scripts/", "/INSTALL_DIR/resource
//-->
</script>
<script type="text/JavaScript">
<!--
   var smartRefreshSupport = true;
//-->
</script>
```

Related Topics

Using Smart-Refresh Technology with Core JSF Components (Section 6.1)

Using Smart-Refresh Technology with NetAdvantage for JSF Components (Section 6.2)

7 Tutorial Programs

This section contains series of step-by-step tutorials that explain key features of the NetAdvantage for JSF components and how to get create projects within the IBM® Rational® Application Developer for WebSphere® 7.0 IDE.

- **IBM RAD Tutorials (Section 7.1)**
- **JSF Component Tutorials (Section 7.2)**

NetAdvantage for JSF

7.1 IBM RAD Tutorials

The following tutorials explain how to create a JSF project within the IBM® Rational® Application Developer for WebSphere® 7.0 IDE and how to bind your project to data in JSF using IBM RAD.

Click on the following links for more information.

- **Create a JSF Project Using IBM RAD (Section 7.1.1)**
- **Data Binding and Method Binding in JSF using IBM RAD (Section 7.1.2)**

7.1.1 Create a JSF Project Using IBM RAD

Before You Begin

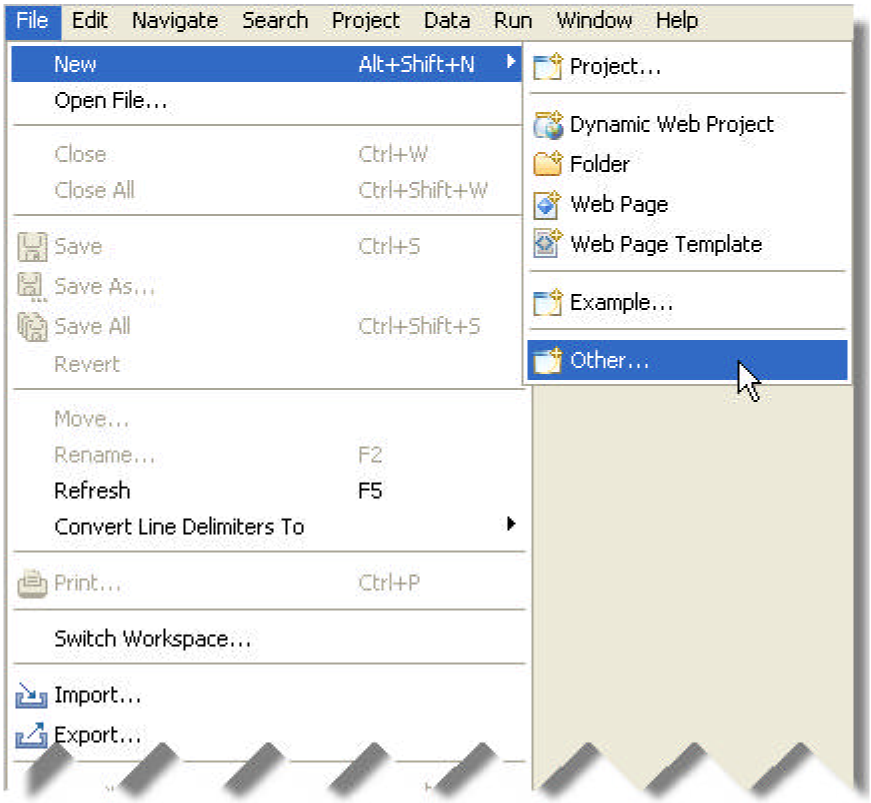
IBM® Rational® Application Developer for WebSphere® 7.0 provides a solid and reliable framework for developing Java Server Faces (JSF) applications.

What You Will Accomplish

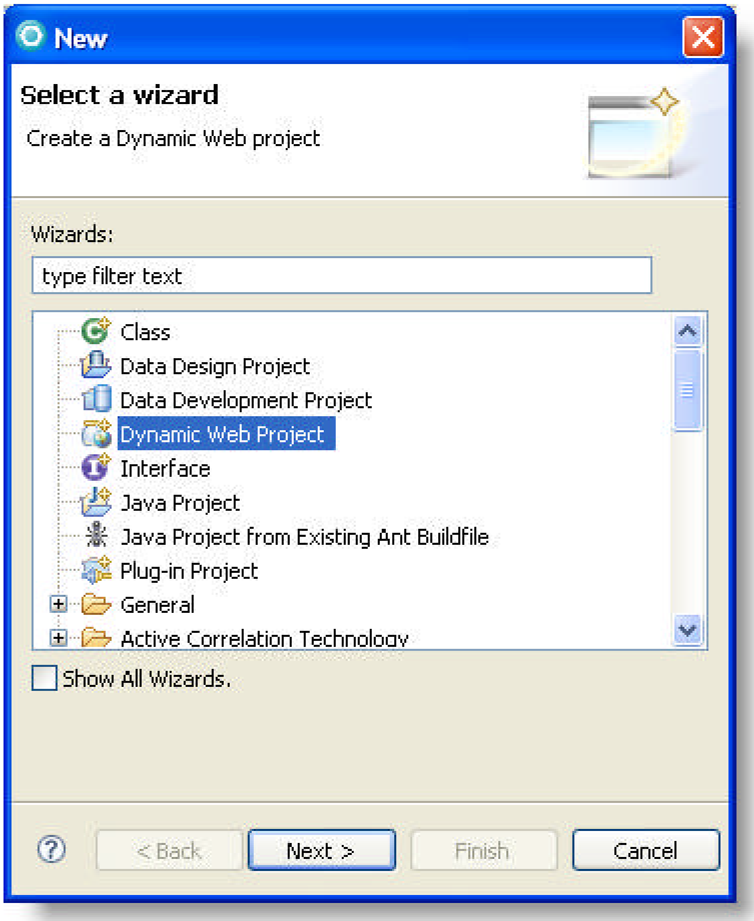
This topic shows you how to create a Java Server Faces (JSF) project using IBM RAD. This topic also draws your attention to the files automatically generated by IBM RAD.

Follow these Steps

1. Open the IBM RAD IDE.
2. On the File menu, point to New and click Other...

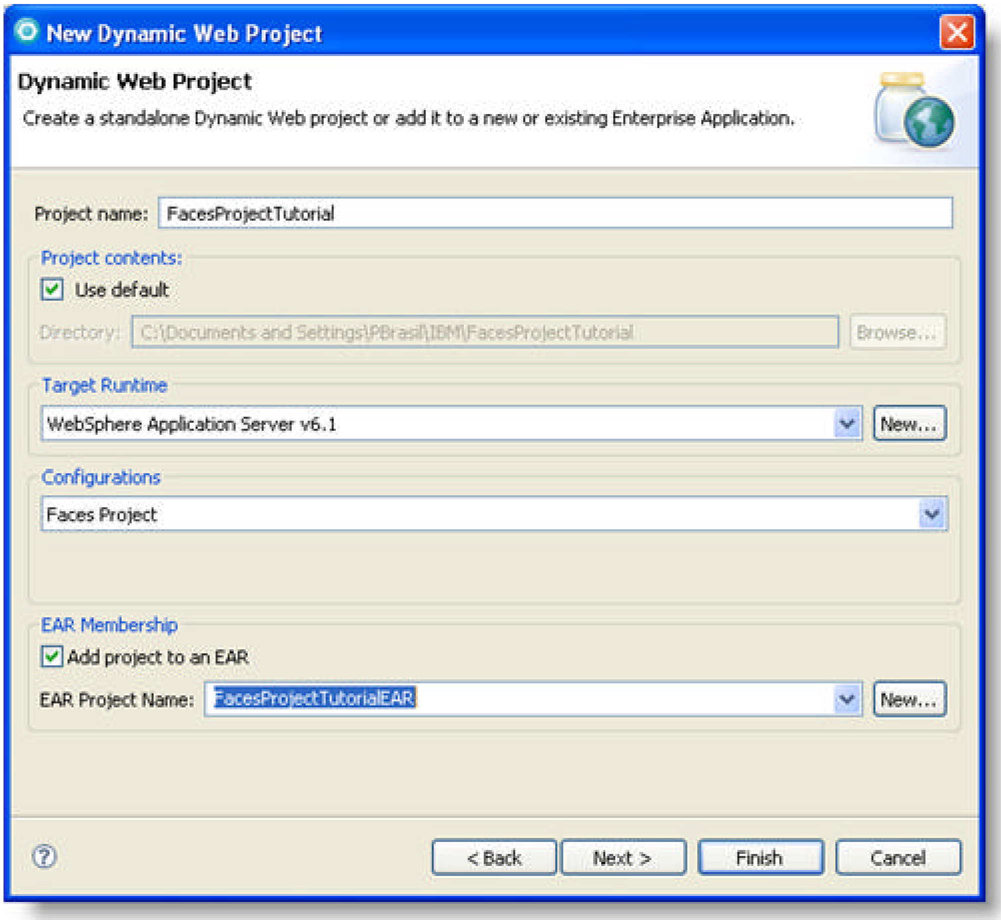


3. In the New wizard, select Dynamic Web Project, then click Next.



4. In the first page of the New Dynamic Web Project wizard, enter the following information, and click Next:

- Project name - Enter a name for the project. In this walkthrough we will name the project FacesProjectTutorial.
- Project contents - If you want the new project to be created in a workspace other than the current one, deselect the "Use default" check box, and click Browse... to navigate to the new workspace. By default, the "Use default" check box is selected.
- Target Runtime - Select a target server for your application. If your server is not listed in the drop-down list, click New... to navigate to a new server.
- Configurations - From the drop-down list, select Faces Project for myfaces support.
- EAR Membership - If you want to add Web components to an enterprise application (EAR file), select the "Add project to an EAR" check box. To select a different EAR project, click New... to browse to the project.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: FacesProjectTutorial

Project contents:

☑ Use default

Directory: C:\Documents and Settings\PBrasil\IBM\FacesProjectTutorial

Target Runtime: WebSphere Application Server v6.1

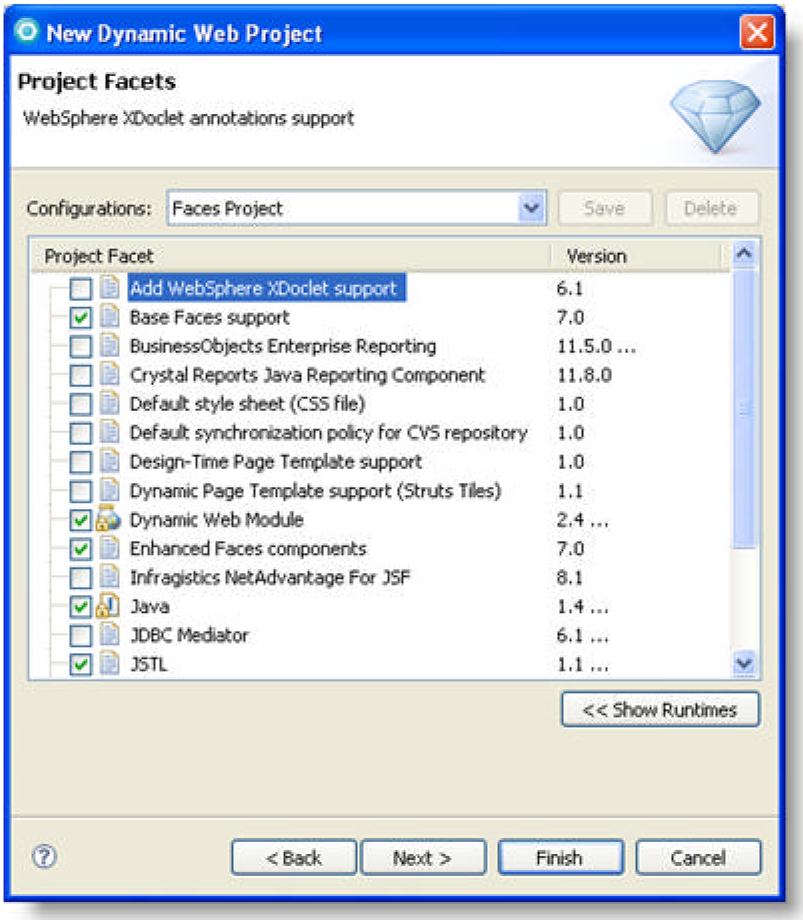
Configurations: Faces Project

EAR Membership

☑ Add project to an EAR

EAR Project Name: FacesProjectTutorialEAR

5. In the Project Facets page of the wizard, ensure that the Base Faces Support and JSTL options are selected. Click Next.



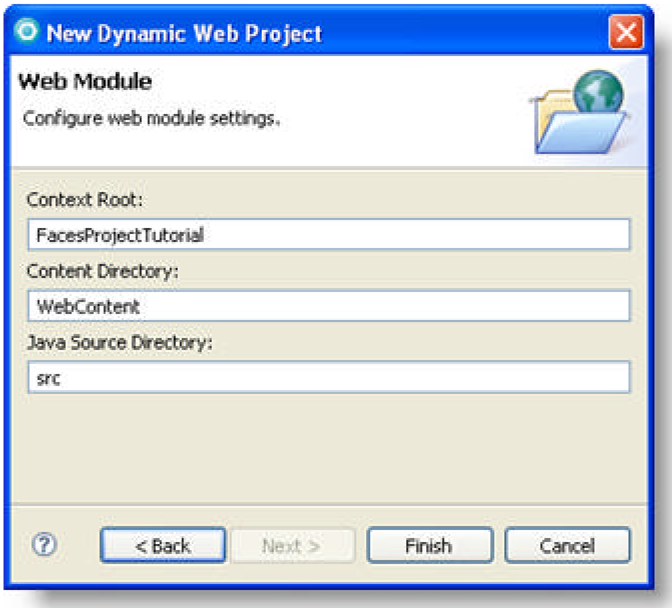
Project Facets

WebSphere XDoclet annotations support

Configurations: Faces Project

Project Facet	Version
□ Add WebSphere XDoclet support	6.1
☑ Base Faces support	7.0
□ BusinessObjects Enterprise Reporting	11.5.0 ...
□ Crystal Reports Java Reporting Component	11.8.0
□ Default style sheet (CSS file)	1.0
□ Default synchronization policy for CVS repository	1.0
□ Design-Time Page Template support	1.0
□ Dynamic Page Template support (Struts Tiles)	1.1
☑ Dynamic Web Module	2.4 ...
☑ Enhanced Faces components	7.0
□ Infragistics NetAdvantage For JSF	8.1
☑ Java	1.4 ...
□ JDBC Mediator	6.1 ...
☑ JSTL	1.1 ...

6. In the Web Module page of the wizard, check that the defaults are acceptable. Click Finish.



Web Module

Configure web module settings.

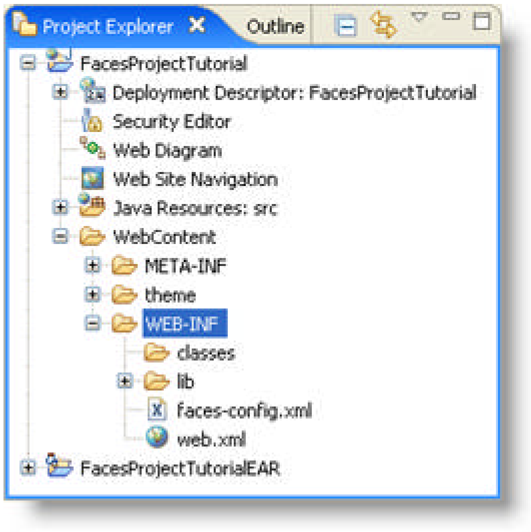
Context Root: FacesProjectTutorial

Content Directory: WebContent

Java Source Directory: src

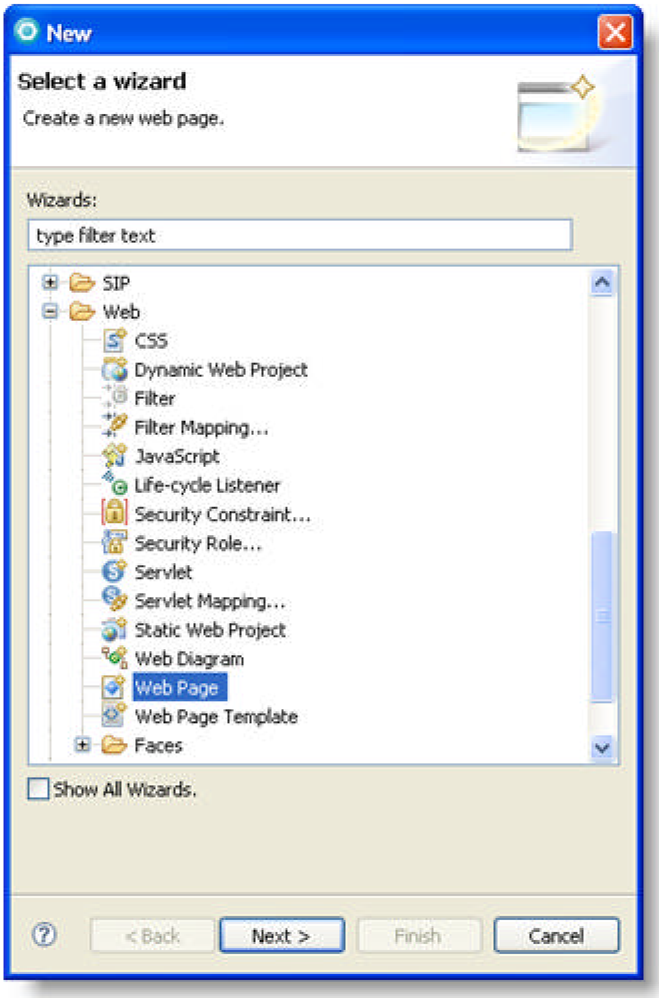
Note: If you are not in the Web Perspective, you will be prompted to switch to it. Click Yes.

7. By default the Web Diagram Editor window appears. Close this file.
8. From the Project Explorer, expand the following nodes:
   - FacesProjectTutorial
   - WebContent
   - WEB-INF



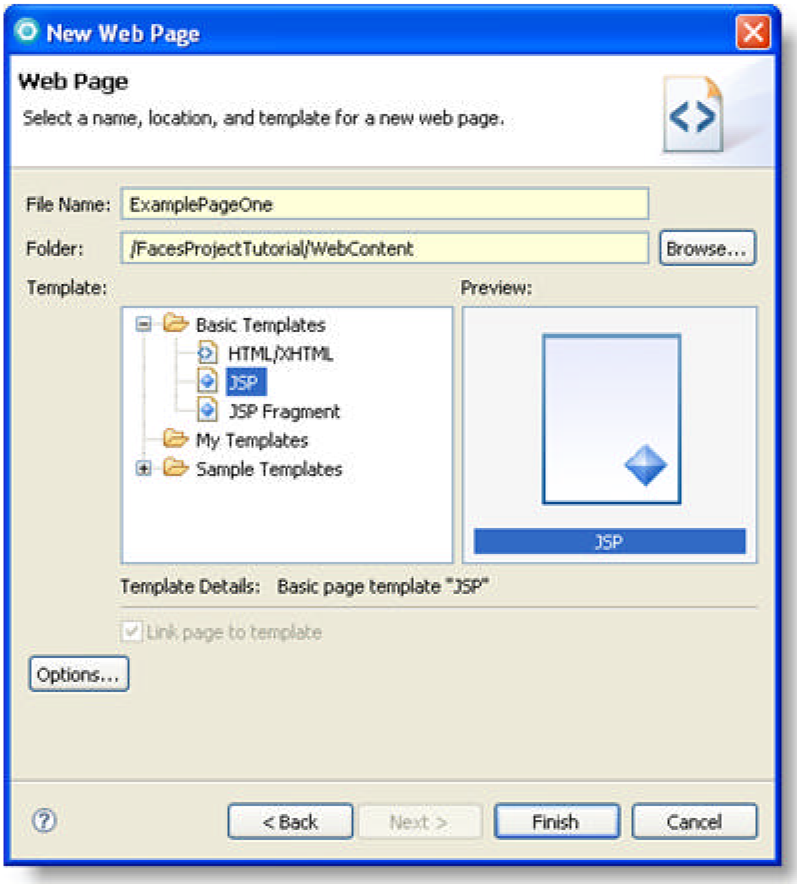
Note: Within the WEB-INF folder, RAD has automatically generated a faces-config.xml file.

9. From the Project Explorer, select the FacesProjectTutorial.
10. On the File menu, point to New and click Other...
11. From the Select a wizard page, expand the Web node and select WebPage.



12. Click Next.

13. In the first page of the New Web Page wizard, enter a name for your page and select the JSP template. In this walkthrough we will name the page, ExamplePageOne.



File Name: ExamplePageOne

Folder: /FacesProjectTutorial/WebContent

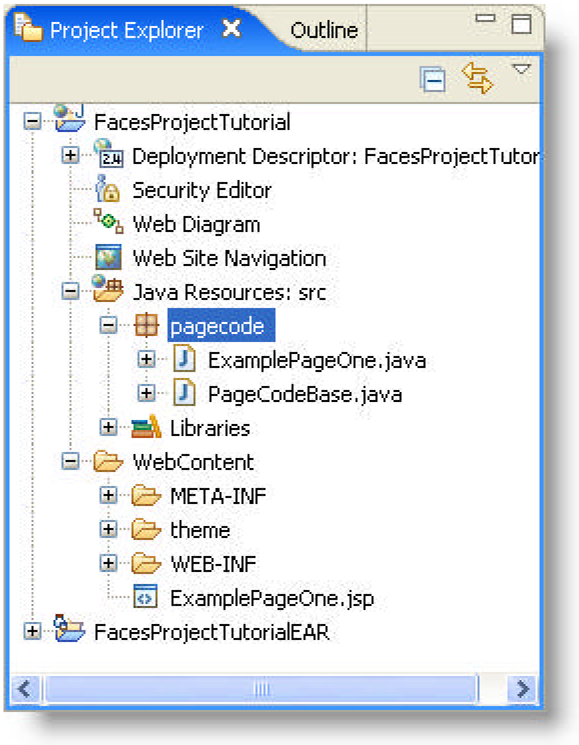
Template Details: Basic page template "JSP"

☑ Link page to template

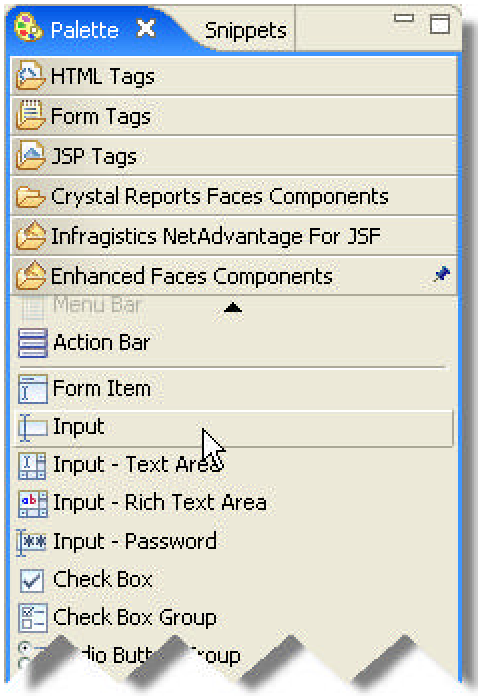
14. Click Finish.
15. From the Project Explorer, expand the following nodes:
    - Java Resources:src
    - pagecode

    There are two java files within the pagecode package, ExamplePageOne.java which is the backing bean for the JSP page we created and PageCodeBase.java which is the base class for all the backing beans generated by RAD.

Note: If you look at the faces-config.xml file you can see that the association between the JSP page and the backing bean is automatically generated.



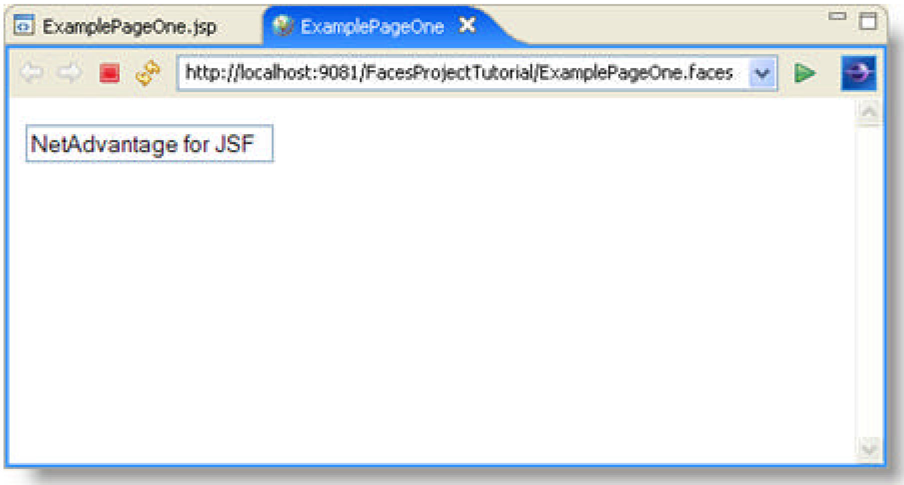
16. Open ExamplePageOne.jsp in design view.
17. From the palette, under the Enhanced Faces Component node, drag the Input component to your page.



18. Open ExamplePageOne.java. As you can see RAD has generated the getter methods. Note that they are protected. The setter methods are not created automatically. If you want to set this component dynamically, you can add the following method.
In Java:

```
public void setText1(HtmlInputText)
{
     text1 = x;
}
```

19. You have successfully created a JSF project using IBM RAD.
20. Save and run your project.



ExamplePageOne.jsp
ExamplePageOne
http://localhost:9081/FacesProjectTutorial/ExamplePageOne.faces

NetAdvantage for JSF

7.1.2 Data Binding and Method Binding in JSF using IBM RAD

Before You Begin

This topic assumes that you already created a JSF project. For information on how to create a JSF project, see **Create a JSF Project Using IBM RAD (Section 7.1.1)**.

What You Will Accomplish

After completing the steps in this tutorial, you will have performed data binding and method binding in a JSF project using IBM® Rational® Application Developer for WebSphere® 7.0. You will have created an input component, radio button component and a drop down menu component that is bound to data.

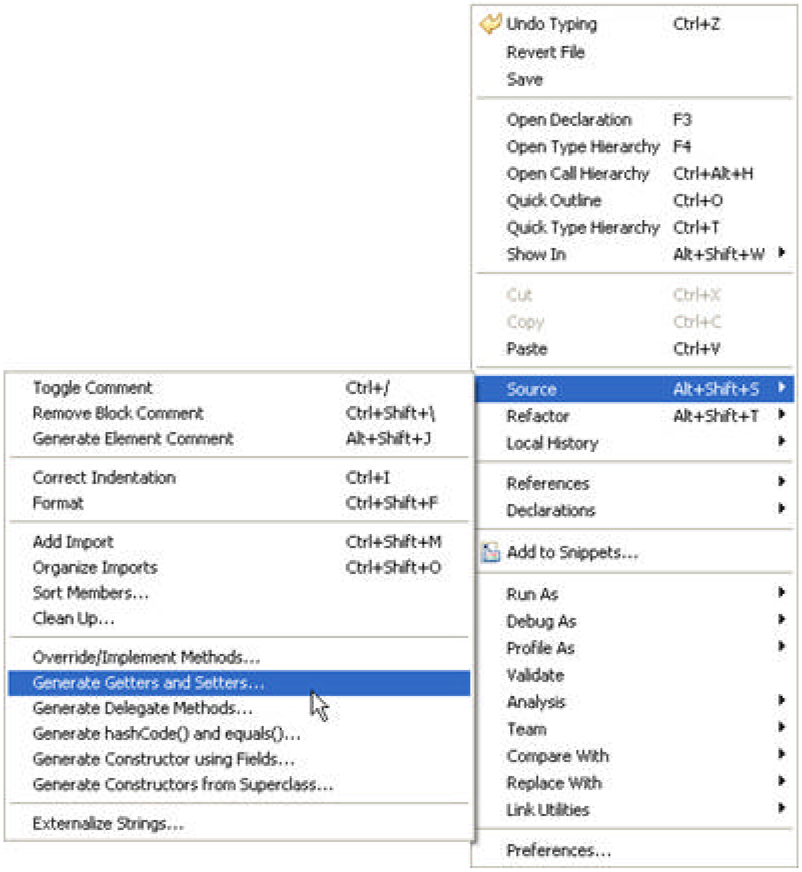
Follow These Steps

1. From the Project Explorer, expand the following nodes:
   - FacesProjectTutorial
   - Java Resources: src
   - pagecode
2. Open ExamplePageOne.java, add the following code:

   In Java:

   ```
   /* Add a new private string called name */
   private String name;
   ```

3. IBM RAD can generate the getter and setter methods. Right click within ExamplePageOne.java. On the context menu, point to Source and select Generate Getters and Setters...



4. In the getName method, hard code the return value to Red.

In Java:

```
/* Set the return value to Red */
public String getName() {
   return "Red";
}
```

Note: The getName() method is used to set the value in the JSP page. The setName() method is used to update the value when the JSP page is submitted.

5. You must then bind the JSP page and the backing bean together. Open the ExamplePageOne.jsp file and set the value attribute of your input component. In the following code, the expression # {pc_ExamplePageOne} binds the input component to the name attribute in the backing bean. pc_ExampleOne refers to an entity in the faces-config.xml file which in turn points to the correct class.

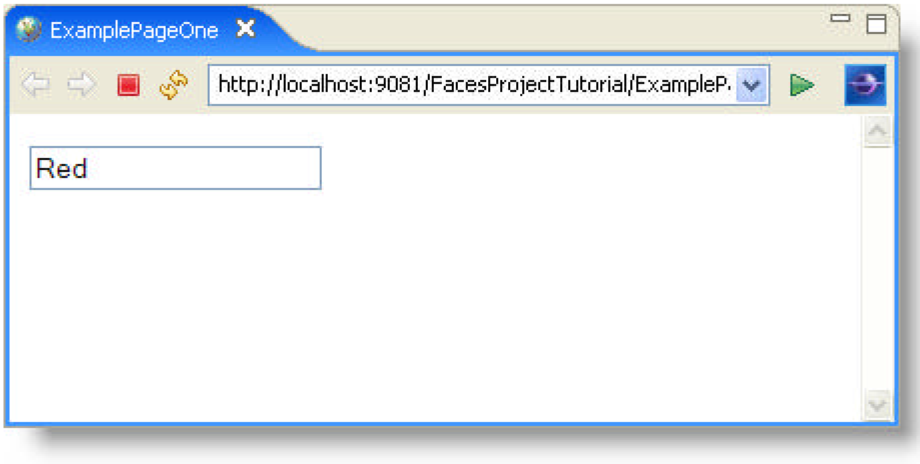
Note: JSF can automatically locate the getter method for the name attribute.

In JSP:

```
<h:inputText
```

```
      id="text1"
      styleClass="inputText"
      value="#{pc_ExamplePageOne.name}">
   </h:inputText>
```

6. Save and run your application. At this point you can see Red displayed within the input component, regardless of what your end user does.

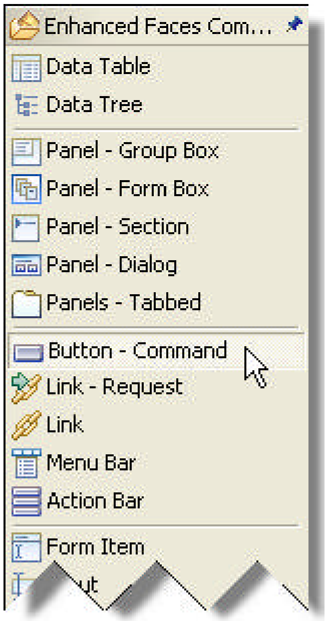


7. The following code changes the getName() method so that it sets the name attribute to Red if the name attribute is null, otherwise it will set the name attribute to the value inputted by your end user.

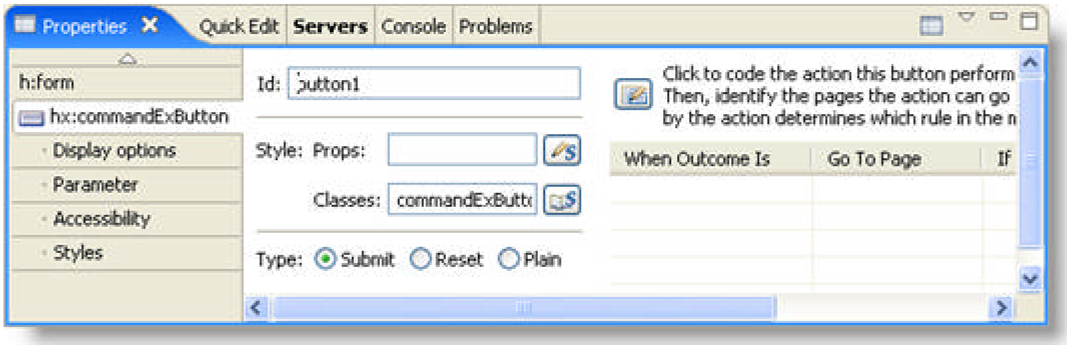
   In Java:

```
public String getName() {
    if(null == name)
       return "Red";
    else
       return name;
}
```

8. Add a submit button to the ExamplePageOne.jsp file. Switch to design view. From the palette, under the Enhanced Faces Component node, drag the Button - Command component to your page.



9. From the Properties window, set the button type to Submit.


Properties ✕ | Quick Edit | Servers | Console | Problems

h:form
hx:commandExButton
Display options
Parameter
Accessibility
Styles

Id: button1

Style: Props:

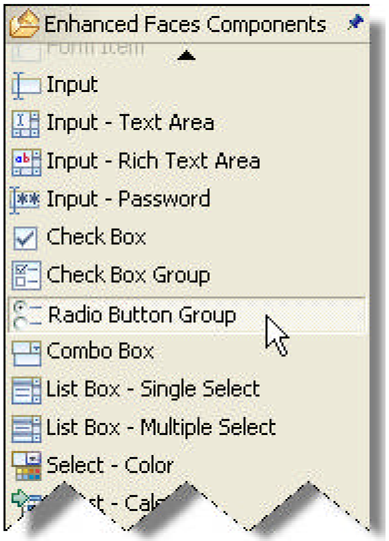
Classes: commandExButt

Type: ● Submit ○ Reset ○ Plain

Click to code the action this button perform
Then, identify the pages the action can go
by the action determines which rule in the n

When Outcome Is	Go To Page	If

10. Add a radio button group to the ExamplePageOne.jsp file. From the palette, under the Enhanced Faces Component node, drag the Radio Button Group component to your page.



11. You must then bind the JSP page and the backing bean together. Open the ExamplePageOne.jsp file and set the value attribute of your radio button group component. In the following code, the expression #{pc_ExamplePageOne.carTypes} binds the radio button group component to the carTypes attribute in the backing bean.

In JSP:

```
<h:selectOneRadio disabledClass="selectOneRadio_Disabled"
        enabledClass="selectOneRadio_Enabled" id="radio1"
        styleClass="selectOneRadio">
        <f:selectItems value=”#{pc_ExamplePageOne.carTypes}” />
</h:selectOneRadio>
```

12. Open the ExamplePageOne.java file, add the following code to add the getCarTypes() method.

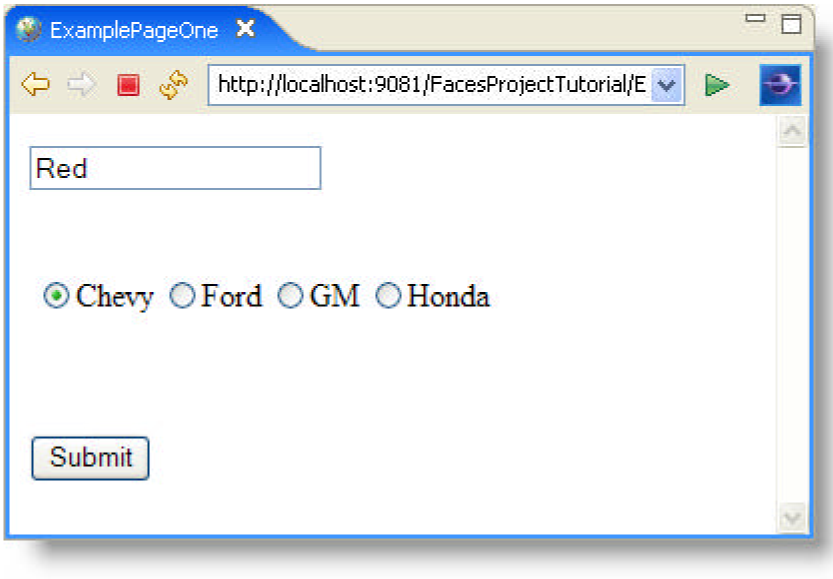
> **Note:** When working with radio buttons you can use a List or a Map to load the data. If you use a List type, you must use objects of type SelectItem. However, this is a JSF specific type and your collection won't be usable outside a JSF context. A more flexible approach is to use a Map type.

In Java:

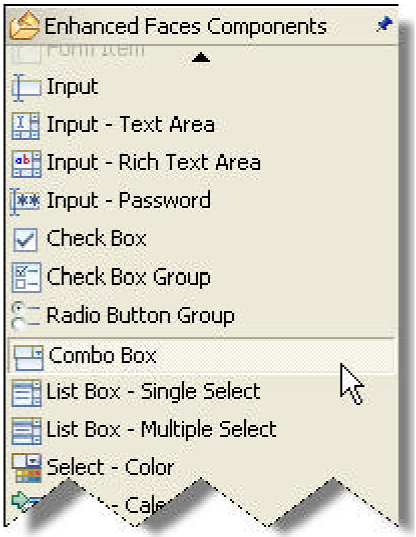
```
Method using List
public ArrayList getCarTypes() {
       ArrayList x = new ArrayList();
       x.add(new SelectItem("Ford"));
       x.add(new SelectItem("Chevy"));
       x.add(new SelectItem("GM"));
       x.add(new SelectItem("Honda"));
       return x;
}

Method using Map
public Map getCarTypes() {
       TreeMap x = new TreeMap();
```

```
            x.put("Ford", "FD");
            x.put("Chevy", "CH");
            x.put("GM", "GM");
            x.put("Honda", "HN");
            return x;
    }
```

13. Save and run your application. Your application should look similar to the screen shot below.


ExamplePageOne
http://localhost:9081/FacesProjectTutorial/E
Red
◉ Chevy ○ Ford ○ GM ○ Honda
Submit

14. Add a radio drop down menu to the ExamplePageOne.jsp file. From the palette, under the Enhanced Faces Component node, drag the Combo Box component to your page.



15. Open the ExamplePageOne.jsp file in Source view. Within the <h:selectOneMenu> tag, add the following code:

In JSP:

```
  <f:selectItems value="#{pc_ExamplePageOne.drivers}" />
```

16. Open the ExamplePageOne.java file and add the following code:
    In Java:

```
public Map getDrivers() {
       TreeMap x = new TreeMap();
       x.put("Joe", "123");
       x.put("Bob", "456");
       x.put("Don", "789");
       x.put("Ray", "321");
       x.put("Sam", "654");
       x.put("Ken", "987");
       return x;
}
```

17. Open the ExamplePageOne.jsp file.

18. Within the <h:selectOneRadio> tag, set the value attribute. This code binds data to the radio button group component so you can save your end users input value.

    In JSP:

```
<h:selectOneRadio disabledClass="selectOneRadio_Disabled"
    enabledClass="selectOneRadio_Enabled" id="radio1"
    styleClass="selectOneRadio"
    value="#{pc_ExamplePageOne.carType}">
```

19. Within the <h:selectOneMenu> tag, set the value attribute. This code binds data to the drop down menu component so you can save your end users input value.

    In JSP:

```
<h:selectOneMenu id="menu1" styleClass="selectOneMenu"
    value="#{pc_ExamplePageOne.driver}">
```

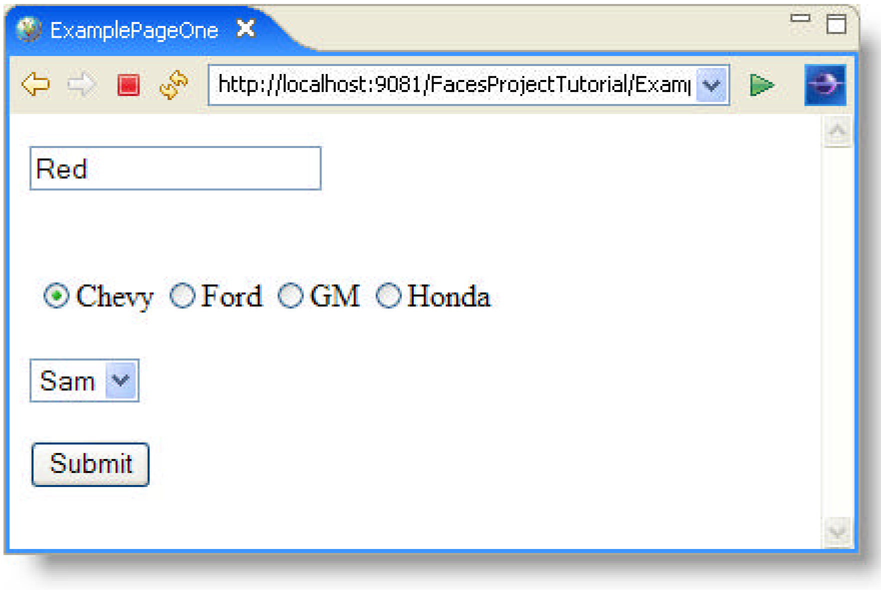
20. Open the ExamplePageOne.java file. Add the following code:

    In Java:

```
private String carType, driver;

public String getCarType() {
    if(carType== null) carType="";
       return carType;
}

public void setCarType(String carType) {
    this.carType = carType;
}

public String getDriver() {
    if(driver == null) driver="";
       return driver;
}
```

```
public void setDriver(String driver) {
    this.driver = driver;
}
```

21. Save and run your application. It should look similar to the screen shot below.



ExamplePageOne
http://localhost:9081/FacesProjectTutorial/Examp
Red
● Chevy ○ Ford ○ GM ○ Honda
Sam
Submit

7.2 JSF Component Tutorials

This section contains series of step-by-step tutorials that explain key features of the NetAdvantage for JSF components.

- **Running the Tutorial Programs (Section 7.2.1)**
- **WebBar (Section 7.2.2)**
- **WebChart (Section 7.2.3)**
- **WebGrid (Section 7.2.4)**
- **WebInput (Section 7.2.5)**
- **IBM RAD Tutorials (Section 7.1)**

7.2.1 Running the Tutorial Programs

All walkthroughs for the project are located in the *install_Dir*/documentation/tutorials folder. For each tutorial, there is a subfolder, as well as a folder for JARs that the tutorials depend upon. Each tutorial contains a build.xml file that can be used to build a WAR file, and deploy it to an Apache® Tomcat server.

Note: The walkthroughs assume that the application server is Tomcat and that the Tomcat server is located in its default install location. In addition, it is assumed that the Infragistics JARs and resource directories are located in the default install location. If you need to deploy the tutorial to an application server other than Tomcat and/or the JARs and resource directories are located in a different location on your system, you must update the build.xml file.

The following is a list of the main tasks in the build.xml file:

- compile -- Compiles all the code in the src directory.
- build_war -- Builds the WAR file for the tutorial program and puts it in the dist directory within the tutorials folder.
- deployToLocalTomcat -- Builds the WAR file and deploys it to a local instance of Tomcat.

Each walkthrough contains a src directory that contains the backing beans, and any other classes that the tutorials depend upon. The config directory contains the JSP™ pages, and a WEB-INF directory that contains the configuration files for the JSF application.

7.2.2 WebBar

In this section, you'll find a step-by-step procedure that walks you through a scenario that is common when using the *WebBar* control:

- **Using WebBar (Section 7.2.2.1)**

7.2.2.1 Using WebBar

The tutorial will show you how to build components at run time as the components are rendered, using WebBar™ components. The components will be created in the Java server-side code of the backing bean, instead of using tags to create the components in the JSP page. This is useful if either the content or the format of the component is not known until run time.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the Install_Dir/documentation/tutorials/DynamicWebBar directory.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.2.1.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.2.1.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.2.1.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.2.1.4)**

7.2.2.1.1 Creating the Backing Bean Part - 1 of 4

First, you need to create the backing bean that will be used to create the SidebarGroups and its sub-items required by the Web page. These groups and items will created in a Java class called DynamicSiderbarGroups.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/DynamicWebBar directory.

A member variable will be used to to access the Sidebar as follows:

In Java:

```
private Sidebar m_bar;
```

The WebBar is modeled using NetAdvantage for JSF's sidebar component and is called m_bar. In order to provide the Web page with access to these objects, you must provide setters and getters that follow the JavaBean standard.

In Java:

```
public Sidebar getSidebar() {
        return m_bar;
}

public void setSidebar(Sidebar bar) {
        m_bar = buildSideBar(bar);
}
```

The sidebar's setter calls the buildSideBar method. This method sets some of the size of the sidebar component, and creates the SidebarGroups.

In Java:

```
private Sidebar buildSideBar(Sidebar bar)
{
  boolean isInitialised = bar.getAttributes().get(IS_INITIALISED)!=null;

  if (!isInitialised) {
    // Get a reference to the JSF Application,
    // we need it to create value bindings later-on
    Application application = FacesContext.getCurrentInstance().getApplication();
    UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();

    /* Create Infragistics SidebarGroup */
    // Set the Sidebar's width and height
    bar.getAttributes().put("style","width: 400px; height: 300px;");
    // Create SidebarGroup
    HTMLSidebarGroup aGroup =
            createInfragisticsSidebarGroup("Infragistics", application, viewRoot);

    // Add the newly created SidebarGroup to the Sidebar's children list
    bar.getChildren().add(aGroup);
    /* Create JSF SidebarGroup */
    // Create SidebarGroup
    aGroup = createJSFSidebarGroup("JSF Information", application, viewRoot);

    // Add the newly created SidebarGroup to the Sidebar's children list
    bar.getChildren().add(aGroup);

    // Set a flag so that we know the tree is initialised
    bar.getAttributes().put(IS_INITIALISED, new Boolean(true));
  }
  return(bar);
}
```

The buildSideBar method makes a call to createInfragisticsSidebarGroup or createJSFSidebarGroup method depending upon which SidebarGroup is being created. Both of these methods have similar functionality by creating a group and populating it with HTMLLinks.

In Java:

```
private HTMLSidebarGroup createInfragisticsSidebarGroup(
          String sidebarGroupNameStr, Application application, UIViewRoot viewRoot)
{
  // Create SidebarGroup, set the Text value, and expand the group
  HTMLSidebarGroup aSidebarGroup = (HTMLSidebarGroup)
           application.createComponent(HTMLSidebarGroup.COMPONENT_TYPE);
  aSidebarGroup.setText(sidebarGroupNameStr);
  aSidebarGroup.setExpanded(true);
  aSidebarGroup.setId(viewRoot.createUniqueId());

  /* Create some Links to display in the group */
  HTMLLink aLink1 = (HTMLLink)application.createComponent(HTMLLink.COMPONENT_TYPE);
  aLink1.setId(viewRoot.createUniqueId());
  aLink1.setURL(
     "http://www.infragistics.com/Corporate/press/PressReleasesViewer.aspx?Id=123");
  aLink1.setValue("Infragistics NetAdvantage JSF Press Release");
  aSidebarGroup.getChildren().add(aLink1);

  HTMLLink aLink2 = (HTMLLink)application.createComponent(HTMLLink.COMPONENT_TYPE);
  aLink2.setId(viewRoot.createUniqueId());
  aLink2.setURL(
       "http://www.infragistics.com/products/NetAdvantage/JSF/Default.aspx");
  aLink2.setValue("Infragistics JSF Home");
  aSidebarGroup.getChildren().add(aLink2);

  HTMLLink aLink3 = (HTMLLink)application.createComponent(HTMLLink.COMPONENT_TYPE);
  aLink3.setId(viewRoot.createUniqueId());
  aLink3.setURL("http://help.infragistics.com/JSF/");
  aLink3.setValue("Infragistics JSF Online Help");
  aSidebarGroup.getChildren().add(aLink3);

  return(aSidebarGroup);
}

private HTMLSidebarGroup createJSFSidebarGroup(String sidebarGroupNameStr,
  Application application, UIViewRoot viewRoot)
{
  //Create SidebarGroup, set the Text value, and expand the group
  HTMLSidebarGroup aSidebarGroup = (HTMLSidebarGroup)
        application.createComponent(HTMLSidebarGroup.COMPONENT_TYPE);
  aSidebarGroup.setText(sidebarGroupNameStr);
  aSidebarGroup.setExpanded(true);
  aSidebarGroup.setId(viewRoot.createUniqueId());

  /* Create some Links to display in the group */
  HTMLLink aLink1 = (HTMLLink)application.createComponent(HTMLLink.COMPONENT_TYPE);
  aLink1.setId(viewRoot.createUniqueId());
  aLink1.setURL(
        "http://www.jsfcentral.com/listings/P11170;
        jsessionid=800015B9EAE2A0BB927D13962BD01340");
  aLink1.setValue("JSF Central - NetAdvantage JSF Press Release");
  aSidebarGroup.getChildren().add(aLink1);

  HTMLLink aLink2 = (HTMLLink)application.createComponent(HTMLLink.COMPONENT_TYPE);
  aLink2.setId(viewRoot.createUniqueId());
  aLink2.setURL("http://www.artima.com/forums/flat.jsp?forum=276&thread=170012");
  aLink2.setValue("Java Community News - NetAdvantage JSF Press Release");
  aSidebarGroup.getChildren().add(aLink2);

  return(aSidebarGroup);
}
```

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.2.1.2)

7.2.2.1.2 Creating the JSP Page - Part 2 of 4

The next step is to create a JSP™ page (dataBind.jsp) that calls the backing bean to get its groups and items that was created in **Creating the Backing Bean - Part 1 of 4 (Section 7.2.2.1.1)**.

> **Note:** The full JSP code, source code, and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/DynamicWebBar directory.

The following code demonstrates how to data bind a WebBar in the JSP file. See the **Code Explanation** section further below for a description of the code.

In JSP:

```
<ig:sidebar
        id="dynGroups"
        binding="#{webbar_dynamic.sidebar}" >
</ig:sidebar>
```

Code Explanation

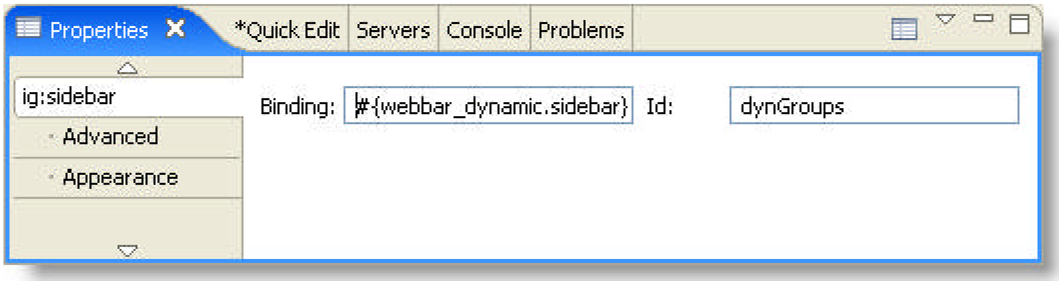
The following is an explanation of the above code.

The <ig:sidebar> tag creates the Sidebar component. This tag has two properties:

- id="dynGroups" -- This property assigns an ID to the Sidebar component, which we can use in the backing bean.
- binding="#{webbar_dynamic.sidebar}" -- This property binds the Sidebar method of the backing bean to the Sidebar component.

To create the Sidebar component at design time using IBM® Rational® Application Developer for WebSphere® 7.0®:

1. From the palette, drag the Sidebar component to your page.
2. In the Properties window, select ig:sidebar.
3. Set the following properties:
    - Binding -- #{webbar_dynamic.sidebar}
    - Id -- dynGroups



Binding: #{webbar_dynamic.sidebar} Id: dynGroups

Related Topic

Updating the Configuration Files - Part 3 of 4 (Section 7.2.2.1.3)

7.2.2.1.3 Configuring the Application - Part 3 of 4

This topic continues from **Creating the JSP Page - Part 2 of 4 (Section 7.2.2.1.2)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the following:

- web.xml
- managed-beans.xml

The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

web.xml

For the web.xml file, the following entries need to be added to enable the Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <URL-pattern>*.faces</URL-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>

    <context-param>
      <param-name>javax.faces.CONFIG_FILES</param-name>
      <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
    </context-param>
```

managed-beans.xml

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the DynamicSiderbarGroups backing bean. To do this, add the entry below.

In XML:

```
<managed-bean>
  <description>
      backing bean used by all samples. Returns a list of employees
  </description>
  <managed-bean-name>webbar_dynamic</managed-bean-name>
```

```
      <managed-bean-class>
          com.infragistics.faces.samples.websidebargroup.DynamicSiderbarGroups
      </managed-bean-class>
      <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.2.1.4)

7.2.2.1.4 Deploying the Application - Part 4 of 4

This topic continues from **Configuring the Application - Part 3 of 4 (Section 7.2.2.1.3)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files. The Apache® Ant build file included in the *Install_Dir*/documentation/tutorials/DynamicWebBar directory contains tasks to create the WAR file and deploy it to an Apache Tomcat server. If your environment is different, you must update the build file.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
    - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the DynamicSiderbarGroups class.
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- dataBind.jsp – The main page that contains the searchable employee grid.
- index.jsp – The home page which simply redirects the user to the dataBind page through the Faces Servlet.

7.2.3 WebChart

In this section, you’ll find step-by-step procedures that walks you through scenarios that are common when using the *WebChart* control:

- **Creating a WebChart (Section 7.2.3.1)**
- **Dynamically Create a WebChart (Section 7.2.3.2)**

7.2.3.1 Creating a WebChart

This section contains a multi-part walkthrough that explains the key features of *WebChart*™.

For the tutorial, three different type of charts will be created; a pie chart, line chart and column chart. The charts will be based on some UN population data that shows the population by continent for the years between 1950 and 2004. This tutorial will be very useful in showing how to associate data with a chart component and how the different attributes of the chart component can be used to produce the desired result.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

Click the links below for access to the multi-part tutorial:

- **Creating the Chart Data (Section 7.2.3.1.1)**
- **Creating the Backing Bean (Section 7.2.3.1.2)**
- **Creating the JSP Pages (Section 7.2.3.1.3)**
- **Configuring the XML (Section 7.2.3.1.4)**
- **Deploying the Application (Section 7.2.3.1.5)**

7.2.3.1.1 Creating Chart Data

Before the backing bean and the JSP pages can be created, the data needs to be formatted in a way that will be understood by the *WebChart*™ component.

The table below shows the data that will be used for the charts throughout this tutorial.

Country Name	1950	1960	1970	1980	1990	2000	2004
Africa	224	282	364	479	636	812	887
Latin America/Caribbean	167	219	285	362	444	523	554
North America	172	204	232	256	283	315	328
Asia	1396	1699	2140	2630	3169	3676	3860
Europe	547	604	656	392	721	728	729
Oceania (Australia, New Zealand, South Pacific)	13	16	20	23	27	31	33

For the different charts, the data needs to be formatted in different ways due to the nature of the chart.

- **Creating Chart Data for Pie Chart (Section 7.2.3.1.1.1)**
- **Creating Chart Data for Line and Column Charts (Section 7.2.3.1.1.2)**

7.2.3.1.1.1 Creating Chart Data for Pie Chart

In this tutorial, the pie chart is used to show the relative population of the continents for a given year. So the data for a pie chart needs to be composed of a series of data points from a specific year (i.e., a column of data from the table).

To create the data for the pie chart a class called PopulationData is created. This class represents the data for a row of the table.

The entire source code for the PopulationData class is available in the following file:

Install_Dir/documentation/tutorial/Chart/src/com/infragistics/samples/chart/PopulationData.java

In the backing bean, a list of PopulationData(m_dataSource) will be created that will contain the data for each continent. By calling the get method for the year you want to show, you will be able to retrieve the data for the pie chart for any given year, as shown in the example below.

In Java:

```
PopulationData popData = new PopulationData("Africa", 224, 282, 364, 479,
636, 812, 887, 2.2, 38.0, 15.0);
m_dataSource.add(popData);

popData = new PopulationData("Latin America & Caribbean", 167, 219, 285,
362, 444, 523, 554, 1.4, 22.0, 6.0);
m_dataSource.add(popData);
```

7.2.3.1.1.2 Creating Chart Data for Line and Column Chart

The column chart and line chart will be used to show the growth of the population of the continents over time. The data for these types of chart requires a series of points for each continent for each year (i.e., a series for each row of the table).

To create the data for the column chart and line chart, a class called PopulationSeriesData is created, which represents a year and population number for a specific country.

The entire source code for the PopulationSeriesData class is available in the following file:

Install_Dir/documentation/tutorial/Chart/src/com/infragistics/samples/chart/PopulationSeriesData.java

In the backing bean a list of PopulationSeriesData is created for each continent. This means that the backing bean will have a get method for each of these populations lists.

In the following example, code for the PopulationSeriesData for Asia will be returned.

In Java:

```
public List getAfricaPopList() {
return m_africaPopList;
}
```

When this data is plotted, the year will be used for the x-axis and the population data will be used for the y-axis.

7.2.3.1.2 Creating the Backing Bean

The entire source code for this backing bean is available in the following file:

Install_Dir/documentation/tutorials/Chart/src/com/infragistics/samples/chart/ChartDataMB.java

The backing bean will use the population data created in the **Creating Chart Data (Section 7.2.3.1.1)** section.

The backing bean is fairly simple because all it has to do is create the data structures needed to create the different charts.

The main method in the backing bean used to create the data for the chart is called createChartData(), which creates the chart data for the pie, column, and line charts.

Besides creating the data for the chart, the backing bean contains a number of getter methods to retrieve the different collections of chart data.

Pie Chart Data

The getDataSource() method retrieves the data list needed for the pie chart.

Line and Column Data

There are also methods for retrieving the data for the column and line charts for each continent. For example, the getAsiaPopList() method retrieves the PopulationSeriesData for Asia. There are similar methods for each continent.

7.2.3.1.3 Creating the JSP Pages

The next step in this tutorial is to create three different JSP pages, one for each chart type.

The JSP pages in this tutorial show simple charts but can be modified and extended fairly easily by changing the different attributes of the WebChart component. For example, by changing the chartType attribute, you can see the different types of charts.

By changing the groupType attribute, you can see how different group types such as Stack, Stack100, Overlay and Grouped can affect each chart type.

To create a combination chart that contains combination chart types, simply specify different chart types for each of the series.

- **Simple Pie Chart (Section 7.2.3.1.3.1)**
- **Grouped Column Chart (Section 7.2.3.1.3.2)**
- **Simple Line Chart (Section 7.2.3.1.3.3)**

7.2.3.1.3.1 Simple Pie Chart

The JSP source code for the pie chart shows how to create a simple 2D pie chart. To make a 3D pie chart, you simply need to change the projectionType attribute to "3D", and add a rotation and perspective attribute.

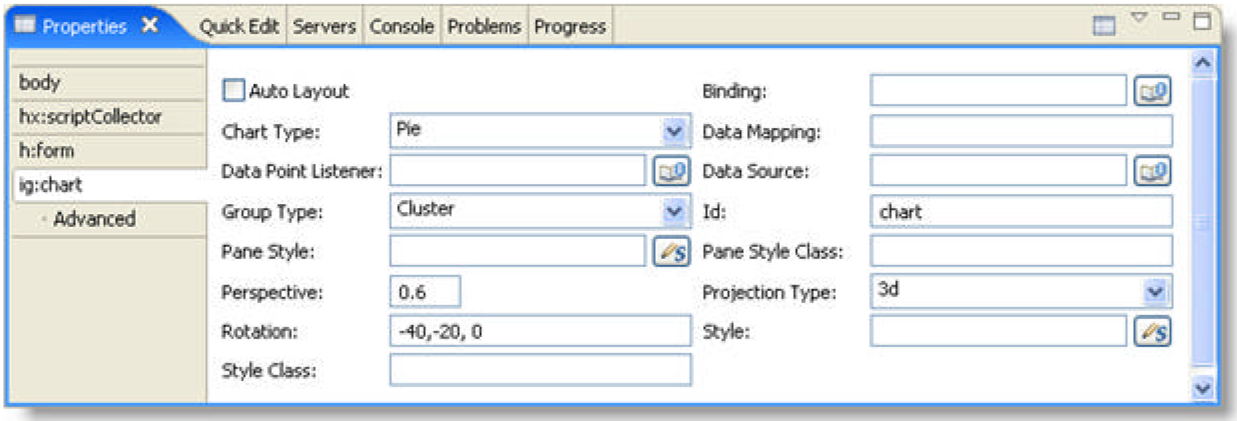
The following code demonstrates how to create a pie chart in the JSP file

In JSP:

```
<ig:chart
    chartType="pie" groupType="cluster" projectionType="3d"
    rotation="-40, -20, 0" perspective="0.6"
    style="width: 520; height: 520">

    <ig:caption caption="Population by Continent 2004"/>
    <ig:series dataMapping="styleClass: style; value: population2004;
        markerCaption: countryName; markerCaptionVisible: captionVisible"
        dataSource="#{chartData.dataSource}" />
</ig:chart>
```

To create a pie chart at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
   - Chart Type -- Pie
   - Group Type -- Cluster
   - Perspective -- 0.6
   - Rotation -- -40, -20, 0
   - Projection Type -- 3d



□ Auto Layout

Chart Type: Pie

Data Point Listener:

Group Type: Cluster

Pane Style:

Perspective: 0.6

Rotation: -40,-20, 0

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

Projection Type: 3d

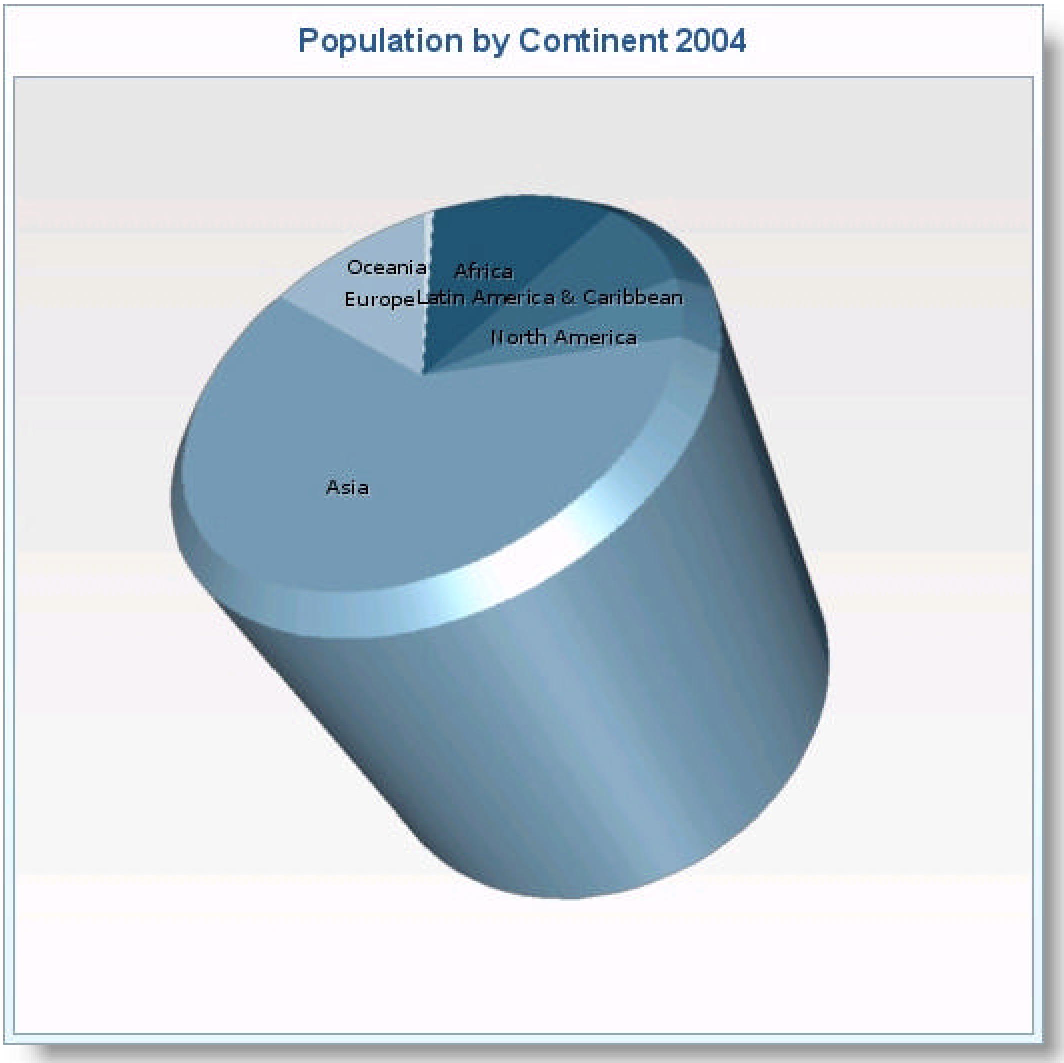
Style:

4. Create other tags in JSP such as <ig:series> and <ig:caption> as they are not components and are not present in the palette.
   In JSP:

```
<ig:caption caption="Population by Continent 2004"/>
```

```
<ig:series dataMapping="styleClass: style; value: population2004;
    markerCaption: countryName; markerCaptionVisible: captionVisible"
    dataSource="#{chartData.dataSource}" />
```

The following screen shot shows a 3D pie chart.



7.2.3.1.3.2 Grouped Column Chart

The JSP code for the column chart shows how to create a simple column chart.

The *WebChart™* component for this chart is similar to the pie chart but with the following differences:

- <ig:axis> -- An x- and y-axis now needs to be defined.
- <ig:series> -- Because this is a group column chart, there needs to be one series specified for each continent.
- chartType -- The chartType is now a column and specified in the series tag.

> **Note:** To reduce the number of attributes specified in the series tag, you can specify chartType attribute in the <ig:chart> tag. Since the series tags are nested inside the <ig:chart> tag the chartType property will default to the value specified in the <chart> tag.

The following code demonstrates how to create a column chart in the JSP file

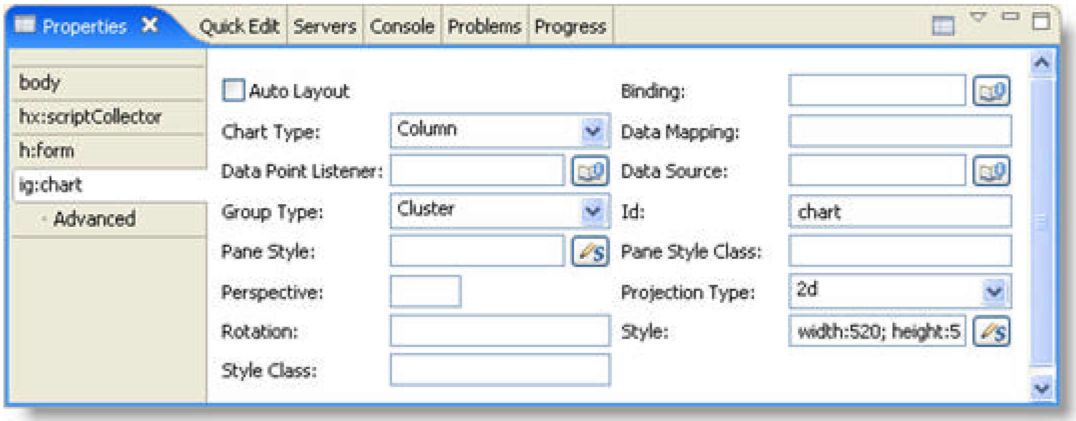
In JSP:

```
<ig:chart
    groupType="cluster" projectionType="2d"
    style="width: 520; height: 520">

    <ig:axis type="x" autoRange="true" autoGridLines="true" />
    <ig:axis type="y" autoRange="true" autoGridLines="true" />

    <ig:caption caption="Population by Continent 1950-2004" position="top"/>
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.asiaPopList}"/>
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.africaPopList}" />
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.laCarPopList}" />
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.namericaPopList}" />
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.europePopList}" />
    <ig:series chartType="column" dataMapping="value: population"
       dataSource="#{chartData.oceaniaPopList}" />
</ig:chart>
```

To create a column chart at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
    - Chart Type -- Column
    - Group Type -- Cluster
    - Projection Type -- 2d

- Style -- width:520; height:520;

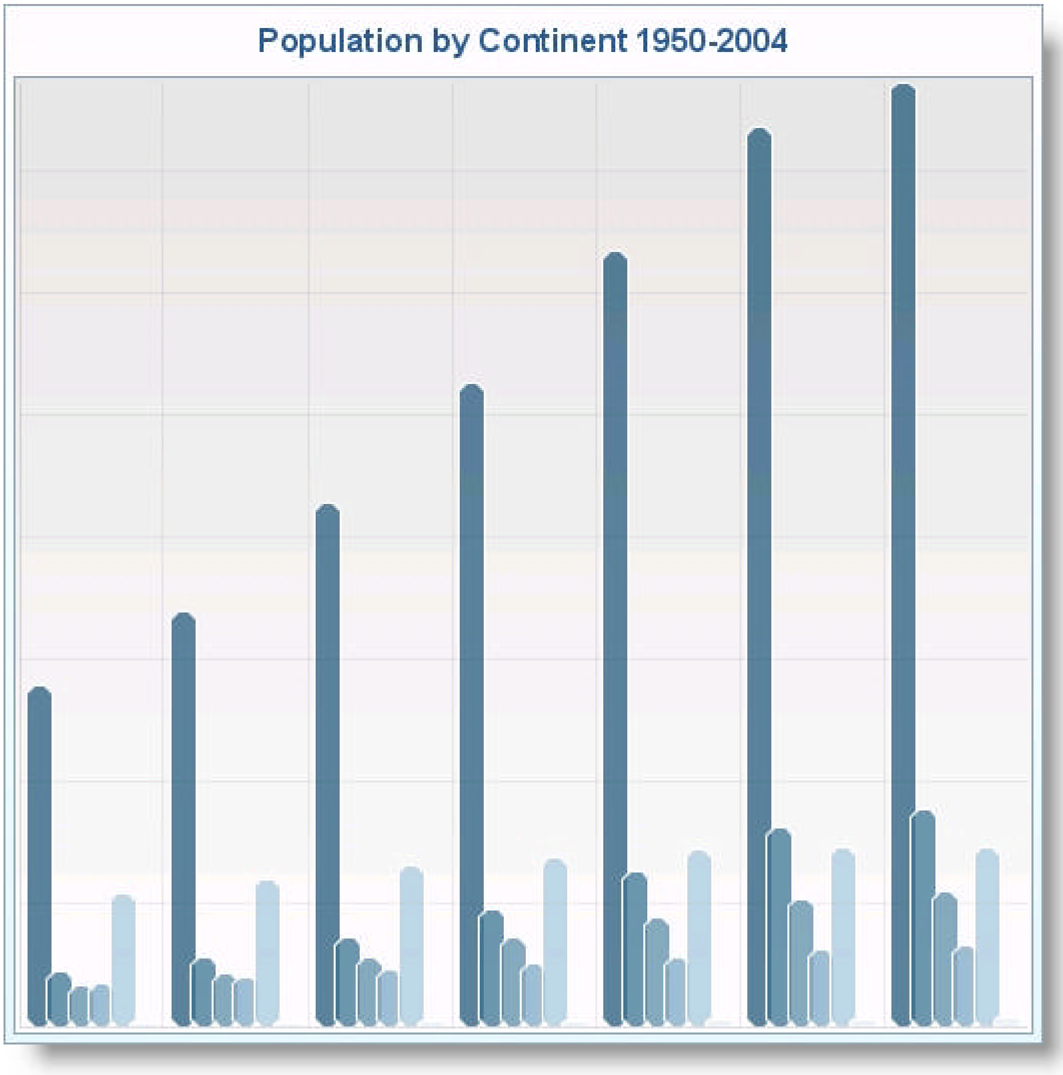


☐ Auto Layout
Chart Type: Column
Data Point Listener:
Group Type: Cluster
Pane Style:
Perspective:
Rotation:
Style Class:
Binding:
Data Mapping:
Data Source:
Id: chart
Pane Style Class:
Projection Type: 2d
Style: width:520; height:5

4. Create other tags in JSP such as <ig:series>, <ig:axis> and <ig:caption> as they are not components and are not present in the palette.
   In JSP:

```
<ig:axis type="x" autoRange="true" autoGridLines="true"/>
<ig:axis type="y" autoRange="true" autoGridLines="true" />
<ig:caption caption="Population by Continent 1950-2004" position="top"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.asiaPopList}"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.africaPopList}"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.laCarPopList}"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.namericaPopList}"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.europePopList}"/>
<ig:series chartType="column"
           dataMapping="value: population"
           dataSource="#{chartData.oceaniaPopList}"/>
```

The following screen shot shows a column chart:


Population by Continent 1950-2004

7.2.3.1.3.3 Simple Line Chart

The JSP code for the line chart is very similar to the column chart. The only real difference is the chartType attribute.

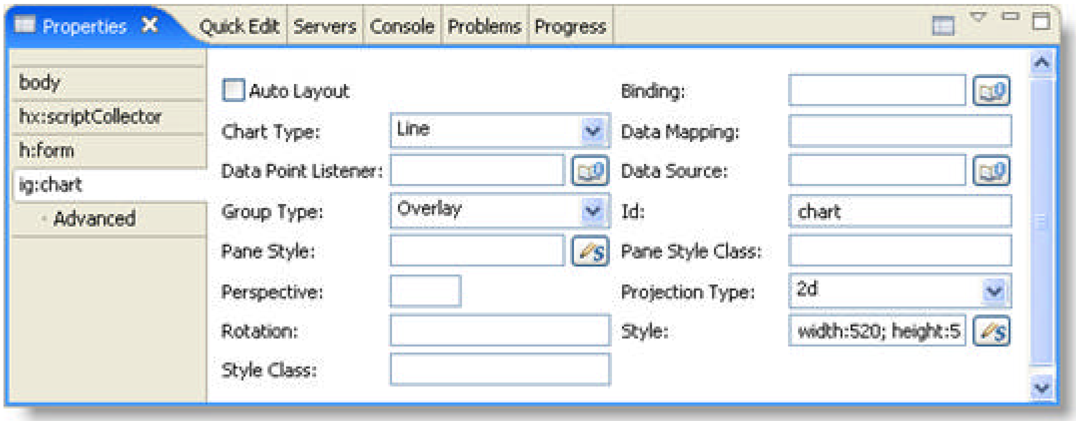
The following code demonstrates how to create a line chart in the JSP file

In JSP:

```
<ig:chart
    groupType="overlay" projectionType="2d" style="width: 520; height: 520">

    <ig:axis type="x" autoRange="true" autoGridLines="true" />
    <ig:axis type="y" autoRange="true" autoGridLines="true" />

    <ig:caption caption="Population by Continent" position="top" />

    <ig:series chartType="area" dataMapping="value: population; "
        dataSource="#{chartData.africaPopList}" />
    <ig:series chartType="area" dataMapping="value: population; "
        dataSource="#{chartData.laCarPopList}" />
    <ig:series chartType="column" dataMapping="value: population; "
        dataSource="#{chartData.namericaPopList}" />
    <ig:series chartType="line" dataMapping="value: population; "
        dataSource="#{chartData.asiaPopList}" />
    <ig:series chartType="line" dataMapping="value: population; "
        dataSource="#{chartData.europePopList}" />
    <ig:series chartType="line" dataMapping="value: population; "
        dataSource="#{chartData.oceaniaPopList}" />
</ig:chart>
```

To create a line chart at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, drag the Chart component to your page.
2. In the Properties window, select ig:chart.
3. Set the following properties:
    - Chart Type -- Line
    - Group Type -- Overlay
    - Projection Type -- 2d
    - Style -- width:520; height:520;



□ Auto Layout

Chart Type: Line

Data Point Listener:

Group Type: Overlay

Pane Style:

Perspective:

Rotation:

Style Class:

Binding:

Data Mapping:

Data Source:

Id: chart

Pane Style Class:

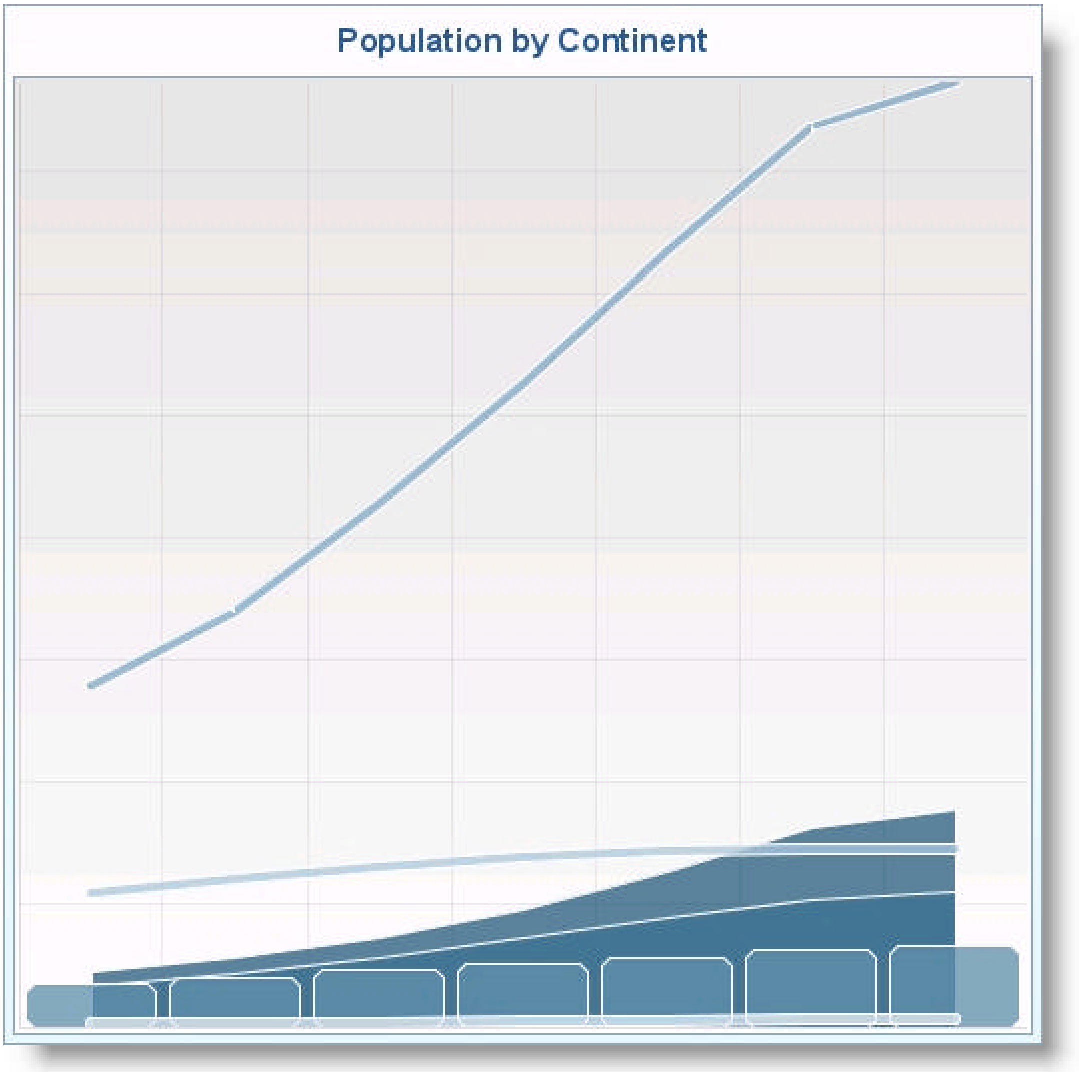
Projection Type: 2d

Style: width:520; height:5

4. Create other tags in JSP such as <ig:series>, <ig:axis> and <ig:caption> as they are not components and are not present in the palette.
   In JSP:

```
<ig:axis type="x" autoRange="true" autoGridLines="true" />
<ig:axis type="y" autoRange="true" autoGridLines="true" />
<ig:caption caption="Population by Continent" position="top" />
<ig:series chartType="area"
           dataMapping="value: population; "
           dataSource="#{chartData.africaPopList}" />
<ig:series chartType="area"
           dataMapping="value: population; "
           dataSource="#{chartData.laCarPopList}" />
<ig:series chartType="column"
           dataMapping="value: population; "
           dataSource="#{chartData.namericaPopList}" />
<ig:series chartType="line"
           dataMapping="value: population; "
           dataSource="#{chartData.asiaPopList}" />
<ig:series chartType="line"
           dataMapping="value: population; "
           dataSource="#{chartData.europePopList}" />
<ig:series chartType="line"
           dataMapping="value: population; "
           dataSource="#{chartData.oceaniaPopList}" />
```

The following screen shot shows a line chart:


Population by Continent

7.2.3.1.4 Configuring the XML

Configuration of this tutorial is simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

The XML files are available in the following folder:

Install_Dir/documents/tutorial/Chart/WEB-INF

For the web.xml file, the following entries need to be added to enable the faces servlet and define various JSF parameters.

In XML:

```
<servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <URL-pattern>*.faces</URL-pattern>
    </servlet-mapping>


    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>
    <context-param>
      <param-name>javax.faces.CONFIG_FILES</param-name>
      <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
   </context-param>
```

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the ChartDataMB backing bean. The following entry needs to be added to the XML file.

In XML:

```
<managed-bean>
   <description>Backing bean for the WebChart</description>
   <managed-bean-name>chartData</managed-bean-name>
   <managed-bean-class>
      com.infragistics.samples.chart.ChartDataMB
   </managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

7.2.3.1.5 Deploying the Application

Deploying the application is basically a matter of putting all the necessary files in the correct location. The ANT build file included in the *Install_Dir/documents/tutorial/Chart* directory contains the tasks needed to create the WAR file and deploy it to an Apache Tomcat server. If your environment is different, then simply update the build file.

The following is a list of the names and locations of the key files:

- resources -- Contains some of the default images used by the controls
    - infragistics -- Contains the JavaScripts files needed by the components.
        - themes -- Contains a set of theme subfolders. Each theme subfolder contain all the CSS style sheets and image files needed by the components. (The default theme is blue.)
- WEB-INF
    - lib -- Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- simplePieChart.jsp --A page with a simple pie chart.
- groupedColumn.jsp -- A page with a grouped column chart.
- simplePieLine.jsp -- A page with a simple line chart.
- index.jsp -- A home page with links to the three chart pages.

7.2.3.2 Dynamically Create a WebChart

This section contains a multi-part walkthrough that explains how to create a WebChart dynamically. For the tutorial, a pie chart will be created. The chart will be based on some UN population data that shows the population by continent for the years between 1950 and 2004.

This tutorial will be very useful in showing how to create a WebChart through the backing bean and how the different attributes of the chart component can be used to produce the desired result.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

Click the links below for access to the multi-part tutorial:

- **Creating the JSP Page (Section 7.2.3.2.1)**
- **Creating the Backing Bean (Section 7.2.3.2.2)**
- **Configuring the XML (Section 7.2.3.2.3)**

7.2.3.2.1 Creating the JSP Page

The JSP source code for the pie chart shows simply binds the chart to your backing bean.

In JSP:

```
<ig:chart
     id = "chartRenderer1"
     binding = "#{pc_ChartPage.chartRenderer1}">
</ig:chart>
```

Related Topic

Creating the Backing Bean (Section 7.2.3.2.2)

7.2.3.2.2 Creating the Backing Bean

The entire source code for this backing bean is available in the following file:

Install_Dir/documentation/tutorials/Chart/src/com/infragistics/samples/chart/ChartDataMB.java

The backing bean will set the properties for the chart.

The main method in the backing bean used to dynamically create the chart is called buildChart().

The following code shows how to create a WebChart in the backing bean.

In Java:

```
//get method for the chart
public HtmlChart getChartRenderer1() {
   if (chartRenderer1 == null) {
      chartRenderer1 = (HtmlChart) findComponentInRoot
      ("chartRenderer1");
   }
   return chartRenderer1;
}

//set method for the chart
public void setChartRenderer1(HtmlChart chartRenderer1) {
   this.chartRenderer1 = buildChart(chartRenderer1);
}

//Method that builds the chart
public HtmlChart buildChart(HtmlChart chart){
   //Set properties on the chart
   chart.setProjectionType("3d");
   chart.setRotation("-50, 0, 0)");
   chart.setPerspective(0.0F);
   chart.setStyle("width:680px;height:350px");
   chart.setGroupType("Overlay");
   chart.setPaneStyle("width:630px;height:350px");

   //Create x Axis
   Axis xAxis = new Axis();
   xAxis.setType(AxisType.X.toString());
   xAxis.setParent(chart);

   //Create y Axis
   Axis yAxis = new Axis();
   yAxis.setType(AxisType.Y.toString());
   yAxis.setParent(chart);

   //Create z Axis
   Axis zAxis = new Axis();
   zAxis.setType(AxisType.Z.toString());
   zAxis.setMinimumValue(-2);
   zAxis.setMaximumValue(2);
   zAxis.setParent(chart);

   //Set chart type and group type
   chart.setChartType(ChartType.PIE.toString());
   chart.setGroupType(SeriesGroupType.Overlay.toString());

   //Create the data for the chart
   Series pieSeries = new Series();
   NumbersSource numbers = new NumbersSource();
   pieSeries.setDataSource(new ListDataModel(ns.getSparse0()));
   pieSeries.setMarkerCaptionVisible(Boolean.FALSE);
   pieSeries.setRadius(new Integer(170));

   pieSeries.setParent(chart);

   //Set the legend properties
   Legend legend = new Legend();
   legend.setAutoItems(true);
   legend.setPosition("RIGHT");
   legend.setStyle("height:350px");
   legend.setParent(chart);

   chart.dataBind(FacesContext.getCurrentInstance());

   return chart;
}
```

Related Topic

Configuring the XML (Section 7.2.3.2.3)

7.2.3.2.3 Configuring the XML

Configuration of this tutorial is simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

The XML files are available in the following folder:

Install_Dir/documents/tutorial/Chart/WEB-INF

For the web.xml file, the following entries need to be added to enable the faces servlet and define various JSF parameters.

In XML:

```
<servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <URL-pattern>*.faces</URL-pattern>
    </servlet-mapping>


    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>
    <context-param>
      <param-name>javax.faces.CONFIG_FILES</param-name>
      <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
   </context-param>
```

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the ChartPage backing bean. The following entry needs to be added to the XML file.

In XML:

```
<managed-bean>
   <description>Backing bean for the WebChart</description>
   <managed-bean-name>chartRenderer1</managed-bean-name>
   <managed-bean-class>
      com.infragistics.samples.chart.ChartPage
   </managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

7.2.3.2.4 Deploying the Application

Deploying the application is basically a matter of putting all the necessary files in the correct location. The ANT build file included in the *Install_Dir/documents/tutorial/Chart* directory contains the tasks needed to create the WAR file and deploy it to an Apache Tomcat server. If your environment is different, then simply update the build file.

The following is a list of the names and locations of the key files:

- resources -- Contains some of the default images used by the controls
  - infragistics -- Contains the JavaScripts files needed by the components.
    - themes -- Contains a set of theme subfolders. Each theme subfolder contain all the CSS style sheets and image files needed by the components. (The default theme is blue.)
- WEB-INF
  - lib -- Contains all the JAR files that this application depends on.
  - web.xml
  - managed-beans.xml
  - navigation.xml
  - faces-config.xml
- ChartPage.jsp

7.2.4 WebGrid

In this section, you'll find step-by-step procedures that walk you through scenarios that are common when using the *WebGrid* control:

- **Connect a WebGrid to a Data Source (Section 7.2.4.1)**
- **Create a Dynamic Hierarchical WebGrid (Section 7.2.4.2)**
- **Create a Fixed Column WebGrid (Section 7.2.4.3)**
- **Create a Hierarchical WebGrid (Section 7.2.4.4)**
- **Dynamically Create a Fixed Column WebGrid (Section 7.2.4.5)**
- **Dynamically Update Data in a WebGrid (Section 7.2.4.6)**
- **Working with Facelets (Section 7.2.4.7)**

7.2.4.1 Connect a WebGrid to a Data Source

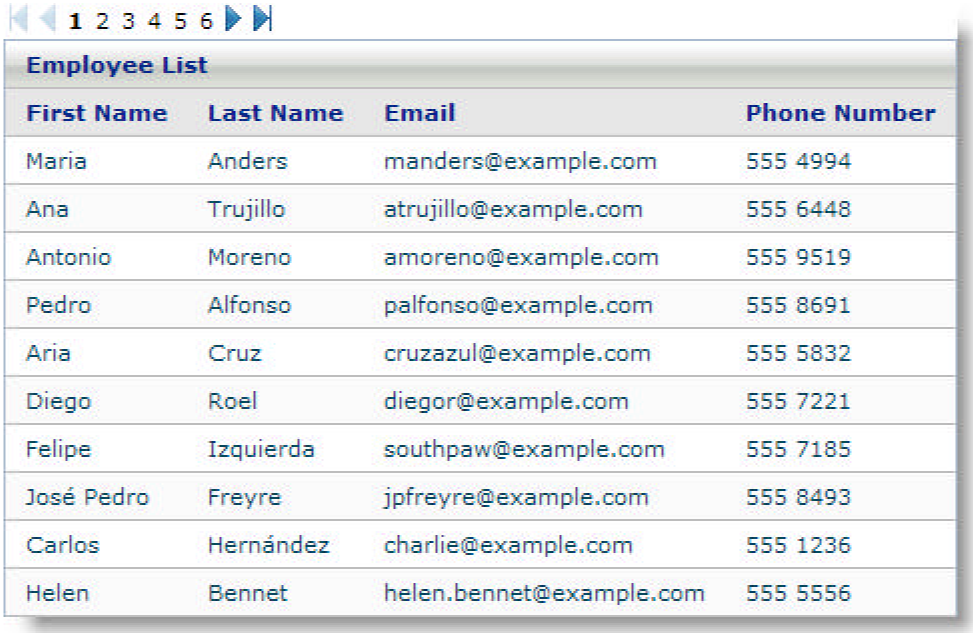
This tutorial will explain the key steps that you need to know to get up and running with *WebGrid*™ components. Although this tutorial is specific to WebGrid, the concepts are valid for most of the NetAdvantage for JSF components.

The goal of this tutorial is to show you how to connect WebGrid to a data source. In the tutorial, the data source will be a simple list of employee data (first name, last name, e-mail address, and phone number).

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/Grid directory.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.1.1)**
- **Creating WebGrid and Binding to a Backing Bean - Part 2 of 4 (Section 7.2.4.1.2)**
- **Updating the Configuration Files - Part 3 of 4 (Section 7.2.4.1.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.4.1.4)**



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

7.2.4.1.1 Creating the Backing Bean - Part 1 of 4

To model the list of employees, we will create a Java class called EmployeeDAO.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/Grid directory.

The data that will be bound to the grid will be retrieved from a List object. In this tutorial, for simplicity purposes, the data for the list is retrieved from a static array in the class. In a real-world application, the data would most likely come from another data source such as a database.

In Java:

```
/** List of employees to be displayed in the grid*/
private static List employees = null;
```

The employees list is simply a list of Employee objects. The Employee class is a JavaBean that consists of fields for the first name, last name, e-mail address, and phone number.

To provide the WebGrid™ components with access to the employee list, you must provide a getter that follows the JavaBean standard.

In Java:

```
public List getEmployees() {
        return employees;
}
```

That is all the code that is needed for the backing bean. The getEmployees() method will be used by the *WebGrid™* to display the data as needed.

Related Topic

Creating WebGrid and Binding to a Backing Bean - Part 2 of 4 (Section 7.2.4.1.2)

7.2.4.1.2 Creating WebGrid and Binding to a Backing Bean - Part 2 of 4

The next step is to create a JSP™ page (dataBind.jsp) that will display a *WebGrid™* component and bind it to the backing bean that was created in **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.1.1)**.

> **Note:** The full JSP code, source code, and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/Grid directory.

The following code demonstrates how to create a WebGrid component in the JSP file. See the Code Explanation section further below for a description of the code.

In JSP:

```
<ig:gridView
        dataSource="#{webgrid_employeeDAO.employees}"
        pagesize="10">
        <f:facet name="header">
                <h:outputText value="Employee List" />
        </f:facet>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="First Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.firstName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Last Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.lastName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Email" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.email}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Phone Number" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.phoneNumber}" />
        </ig:column>
</ig:gridView>
```

Code Explanation

The following is a list of the steps that occur in the above code.

1. The first tag ig:gridView creates the WebGrid component. This tag has two properties

    - dataSource= "#{webgrid_employeeDAO.employees} -- This property binds the Employees list of the backing bean to the WebGrid component.

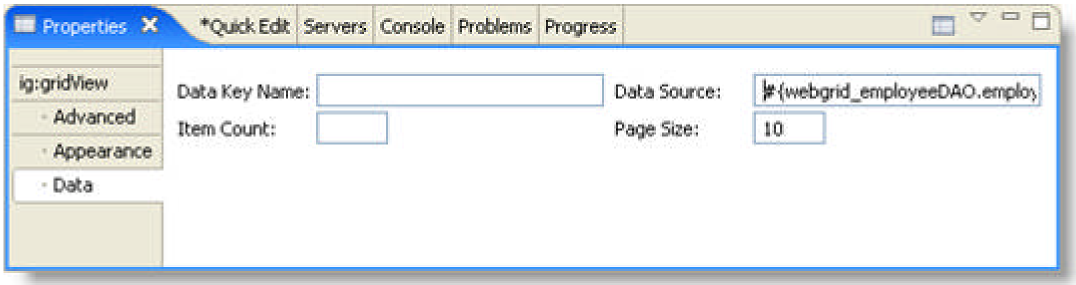
- pagesize="10" -- This property indicates to the WebGrid component to display 10 rows per page.

3. The next block of code specifies a header area in WebGrid where you can place a title for the data being shown.
4. The next block of code is a series of ig:column tags that are used to define the column header and data value for each column of the WebGrid component. For this example, the data value for each column is a simple outputText component that is bound to one of the fields of the Employee object. The DATA_ROW part of the value binding is a built-in constant that is used to enumerate the rows of the grid when it is rendered. See **Identify the Rows in a Collection (Section 4.7.3.1.5)** for more information.

Note: The WebGrid component is simply a container so the data value for a column can be represented by any type of component, including drop-down lists, check boxes, inputText fields, etc.

To create WebGrid in design view using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, drag the GridView component to your page.
2. In the Properties window, under ig:gridView, select Data .
3. Set the following properties:
   - Data Source: #{webgrid_employeeDAO.employees}
   - Page Size: 10



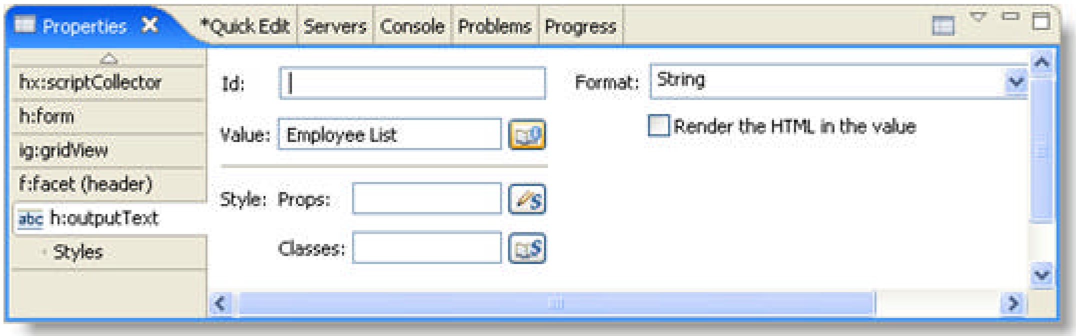
Data Key Name:

Data Source: #{webgrid_employeeDAO.employ

Item Count:

Page Size: 10

4. In the Design view, select the Grid Header and set the following property:
   - Value -- Employee List



Id:

Value: Employee List

Format: String

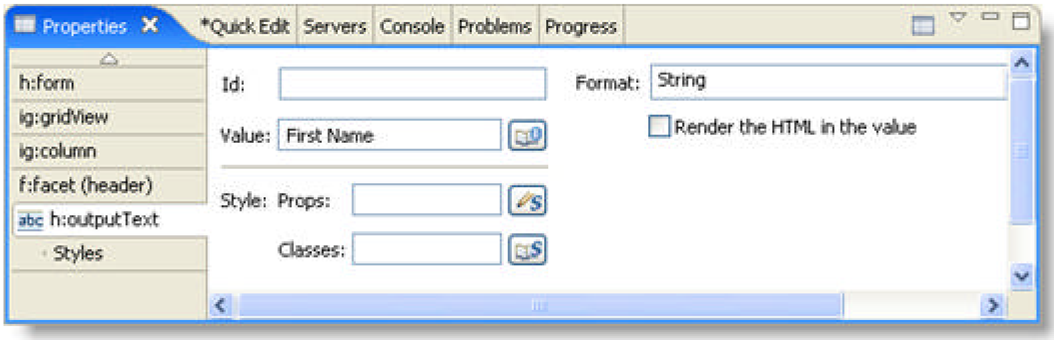
□ Render the HTML in the value

Style: Props:

Classes:

5. The default GridView is created with two columns. From the palette, drag the Column component to your grid to add more columns.
6. In the Design view, for each column header, set the Value property:

- For Column Header 1, set Value to First Name.
- For Column Header 2, set Value to Last Name.
- For Column Header 3, set Value to Email.
- For Column Header 4, set Value to Phone Number.

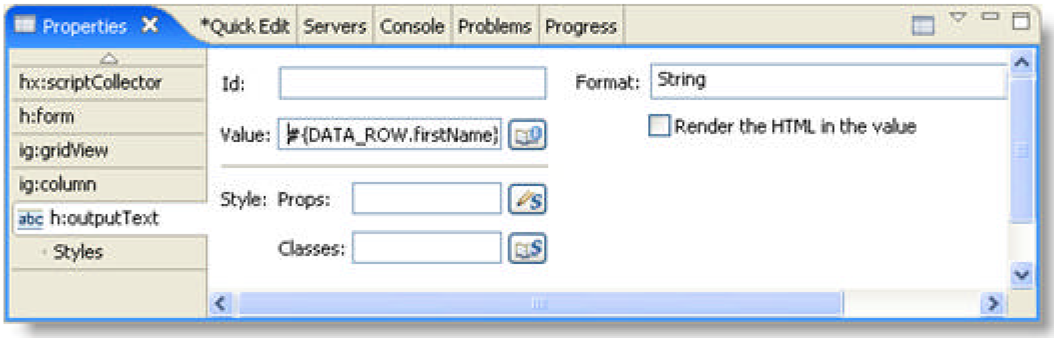


Id:

Value: First Name

Format: String

7. In the Design view, for each column, set the Value property:
   - For Column 1, set Value to #{DATA_ROW.firstName}.
   - For Column 2, set Value to #{DATA_ROW.lastName}.
   - For Column 3, set Value to #{DATA_ROW.email}.
   - For Column 4, set Value to #{DATA_ROW.phoneNumber}.



Id:

Value: #{DATA_ROW.firstName}

Format: String

Related Topic

Updating the Configuration Files - Part 3 of 4 (Section 7.2.4.1.3)

7.2.4.1.3 Updating the Configuration Files - Part 3 of 4

This topic continues from **Creating WebGrid and Binding to a Backing Bean - Part 2 of 4 (Section 7.2.4.1.2)**.

Configuration of this JSF application is relatively simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed. For the web.xml file, you must add the following entries to enable the Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <URL-pattern>*.faces</URL-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>

    <context-param>
      <param-name>javax.faces.CONFIG_FILES</param-name>
      <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
    </context-param>
```

The managed-beans.xml file needs to be updated to define the connection between the alias used in the JSP file and the EmployeeDAO backing bean. To do this, we add the entry below.

In XML:

```
<managed-bean>
        <description>Returns a list of employees</description>
        <managed-bean-name>webgrid_employeeDAO</managed-bean-name>
        <managed-bean-class>com.infragistics.faces.samples.data.EmployeeDAO
        </managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.4.1.4)

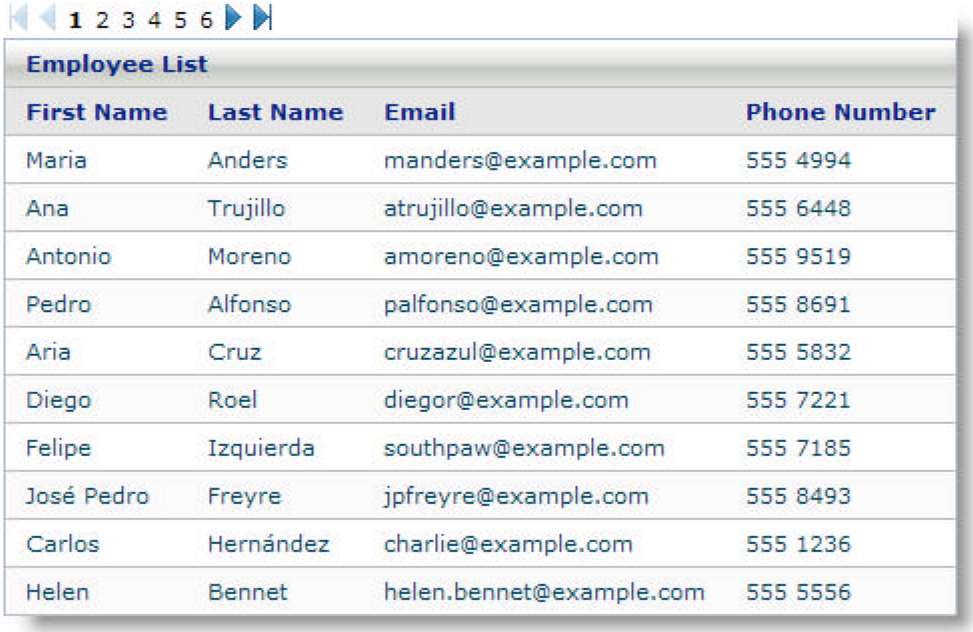
7.2.4.1.4 Deploying the Application - Part 4 of 4

This topic continues from **Updating the Configuration Files - Part 3 of 4 (Section 7.2.4.1.3)**.

Deploying the application is simply a matter of building the WAR file with all the pieces in the correct location. Below is a brief description of the names and locations of the key files. The Apache® Ant build file that is included in the *Install_Dir*/documentation/tutorials/Grid directory contains tasks to create the WAR file and deploy it to an Apache Tomcat server. If your environment differs, simply update the build file.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
  - /lib -- Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
  - web.xml
  - managed-beans.xml
  - navigation.xml
  - faces-config.xml
- dataBind.jsp -- The main page that contains the searchable employee grid.
- index.jsp -- The home page which simply redirects the user to the dataBind page through the Faces Servlet.

When you run the application, WebGrid will appear similar to the following screen shot.



Employee List

First Name	Last Name	Email	Phone Number
Maria	Anders	manders@example.com	555 4994
Ana	Trujillo	atrujillo@example.com	555 6448
Antonio	Moreno	amoreno@example.com	555 9519
Pedro	Alfonso	palfonso@example.com	555 8691
Aria	Cruz	cruzazul@example.com	555 5832
Diego	Roel	diegor@example.com	555 7221
Felipe	Izquierda	southpaw@example.com	555 7185
José Pedro	Freyre	jpfreyre@example.com	555 8493
Carlos	Hernández	charlie@example.com	555 1236
Helen	Bennet	helen.bennet@example.com	555 5556

7.2.4.2 Create a Dynamic Hierarchical WebGrid

Using the MasterDetail facet in a column of the grid on a JSP™ page will allow you to create a hierarchical grid that will contain your data. However, this assumes that the hierarchy of the data is static and known when the page is being designed. If data is more dynamic, and the hierarchy can change based on user input, then coding the hierarchy using the JSP tags will not work. This tutorial will show you how to create a dynamic hierarchical grid that can change based on user input.

For this tutorial, we are going to create a hierarchical grid to display information about a playlist of songs. The list will be arranged in either of the following orders based on user input:

- Artist > Year > Song
- Year > Artist > Song

The page will contain a drop-down list to allow the end user to select between the different ways to display the hierarchical data. It will also contain a hierarchical grid comprised of three levels.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/Dynamic HGrid directory.

The Dynamic Hierarchical Grid tutorial is a multi-part tutorial:

- **Creating the Data Structure for the Playlist - Part 1 of 5 (Section 7.2.4.2.1)**
- **Creating the Backing Bean - Part 2 of 5 (Section 7.2.4.2.2)**
- **Creating the JSP Page - Part 3 of 5 (Section 7.2.4.2.3)**
- **Configuring the Application - Part 4 of 5 (Section 7.2.4.2.4)**
- **Deploying the Application - Part 5 of 5 (Section 7.2.4.2.5)**

7.2.4.2.1 Creating the Data Structure for the Play list - Part 1 of 5

Before we can create the backing bean and JSP™ page, we need to create the hierarchical data structure for our play list data.

In the source code for this example, there are a number of classes in the com.infragistics.faces.samples.hgrid package that are used to define data structure. The MusicData class is the main data class. This class is used to contain information about each of the songs in the play list. The MusicData class contains fields for the song name, artist name, album title, song length, etc.

The MusicData class also contains a field called "sublist". The sublist field is meant to be a link to a list of other music data. The data for the hierarchical grid is simply a set of nested lists of music data. For example, if the end user wants to display the playlist data by Artist > Year >Song, you would perform the following steps to create the data needed for the hierarchical grid:

1. Create an artist list of music data containing a list of unique artists.
2. For each entry in the artist list, create a year sub-list of music data containing the years of the artist's songs.
3. Add the year sub-list to the artist list entry.
4. For each entry in the year sub-list, create a song sub-list of music data containing song data for the year for that artist.
5. Add this song sub-list to the year list entry.

The music data is read from the MyMusic.csv file. The PlayListReader class contains a readPlayList that reads all the song information from the file and converts it into a list of music data. The PlayList class contains a set of utility methods to filter a set of music data by artist or year and create the sub-lists needed to create the different hierarchies. The HierMusicData class contains two public methods to create the lists needed by the hierarchical grid to display the music data by Artist > Year >Song, or Year > Artist > Song.

For a complete understanding of the playlist data structure, refer to the source code and unit tests for this tutorial.

Related Topic

Creating the Backing Bean - Part 2 of 5 (Section 7.2.4.2.2)

7.2.4.2.2 Creating the Backing Bean - Part 2 of 5

The backing bean for this tutorial will use the play list data structure to dynamically create the hierarchical grid.

The Web page will contain two main components:

- drop-down list to allow the end user to choose how to display the music data
- hierarchical grid to display the music data

The drop-down list value is modeled using a standard JSF UIInput class and is called listType. The hierarchical grid is modeled using the Infragistics GridView component and is called m_hgrid.

In Java:

```
/** Grid used to show music data*/
private GridView m_hgrid;
/** value used to determine which type of hierarchy to create */
UIInput listType = null;
```

The hierarchical grid is created dynamically by the buildGrid method. This method is called each time the setHGrid method is called.

In Java:

```
/**
* Dynamically builds the hierarchical grid
*
* @param grid
* @return
*/
private GridView buildGrid(GridView grid) {
 System.out.println("building Grid");

        if(m_musicListType.equals("year"))
        {
                grid = buildYearGrid(grid);
        }
        else if(m_musicListType.equals("artist"))
        {
                grid = buildArtistGrid(grid);
        }
        return grid;
}
```

Based on which type of grid the end user wants to display, this method will call either the buildYearGrid() or buildArtistGrid() method.

The buildArtistGrid() method builds the hierarchical grid to display the Artist > Year > Song data. It does this by doing the following

1. Clears the current grid clearGrid();
2. Creates a new set of music data from the csv file createMusicData();

3. Binds the grid to the music data
4. Creates a single column in the grid to show the artist name createArtistColumn()

The createArtistColumn() method creates the artist column and the nested sub-grid to show the year data for this artist. The nested sub-grid is created by the createNestedYearGrid() method. This method creates a grid with one column to show the years for the artist's songs. This grid also has a sub-grid of the song for that year for that artist. This sub-grid is create by createNestedSongGrid() method.

In Java:

```
/**
* Creates a top level column to display the artist name from the music list
* It also creates a nested grid showing all the years of the songs by
* each artist
*
* @param colNameStr
* @param valueBindingStr
* @param application
* @param viewRoot
* @return
*/
private Column createArtistColumn(String colNameStr,
  String valueBindingStr,
  Application application,
  UIViewRoot viewRoot) {
        Column aColumn = (Column) application
          .createComponent(Column.COMPONENT_TYPE);
        GridView nestedGrid = createNestedYearGrid(application, viewRoot);
        aColumn.setMasterDetail(nestedGrid);
        // Create the col header name value
        UIOutput colName = (UIOutput) application
          .createComponent(UIOutput.COMPONENT_TYPE);
        colName.setValue(colNameStr);
        // colName.setId(viewRoot.createUniqueId());

        // Create the output text tag used to show the value of the column
        HTMLOutputText colValue = (HTMLOutputText) application
          .createComponent(HTMLOutputText.COMPONENT_TYPE);
        colValue.setValueBinding("value", application
          .createValueBinding(valueBindingStr));

        // set the header for the column
        aColumn.setHeader(colName);
        // aColumn.setHeader(headerMenu);
        aColumn.getAttributes();

        // Add the HTML output text field to show the column value for each row
        aColumn.getChildren().add(colValue);

        return (aColumn);
}
```

The createNestedYearGrid() method returns a GridView component. To create the nested grid in the artist column, you simply need to call the column's setMasterDetail() method passing the GridView component created by the createNestedYearGrid() method. The rest of the createArtistColumn() method creates the template for the column header and value to display in the column.

The buildYearGrid() method follows a similar pattern to the buildArtistGrid() method except that it creates the hierarchical grid in a different order using the createYearColumn() and createNestedArtistGrid() methods.

The drop-down list that is used to change the type of hierarchical grid is hooked up to a valueChangeListener onChange() in the backing bean. Each time the users selects an item in the drop-down list, this method gets called.

In Java:

```
public void onChange(ValueChangeEvent event) {
  System.out.println("Drop Down changed");
        m_musicListType = listType.getValue().toString();
        createMusicData();
        m_hgrid.getAttributes().put(IS_INITIALISED,
          new Boolean(false));
        buildGrid(m_hgrid);
}
```

This method does the following:

1. Gets the value chosen from the drop-down list and sets the musicListType parameter.
2. Recreates the music data to match the format of the hierarchy selected.
3. Sets the IS_INITIALISED string to False so the grid will reinitialize itself.
4. Calls the buildGrid() method to rebuild the grid.

For more details on the methods of the backing bean, refer to the source code for this tutorial.

Related Topic

Creating the JSP Page - Part 3 of 5 (Section 7.2.4.2.3)

7.2.4.2.3 Creating the JSP Page - Part 3 of 5

The next step is to create a JSP™ page (hgrid.jsp) that will display the components and bind the values and actions to the backing bean.

Note: The full JSP code, source code and resources needed to build and deploy this tutorial can be found in the Install_Dir/documents/tutorial/Dynamic HGrid directory.

Besides the standard HTML and tags, the JSP contains the tags for the drop-down list and GridView component.

In JSP:

```
<f:view>
  <h:form>
    <h:panelGroup styleclass="main" >
      <h:panelGroup styleclass="section">
        <h:panelGrid columns="2" >
          <h:outputText value="Select How to Order Hierarchy: " />
          <ig:dropDownList id="listView"
                      dataSource="#{musicBean.listViewType}"
                      binding= "#{musicBean.listType}"
                      valueChangeListener="#{musicBean.onChange}"
                      smartRefreshIds="musicGrid" />
        </h:panelGrid>
      </h:panelGroup>
      <h:panelGroup styleclass="section">
        <ig:gridView id="musicGrid" binding="#{musicBean.hgrid}" />
      </h:panelGroup>
    </h:panelGroup>
  </h:form>
</f:view>
```

The drop-down list is an Infragistics drop-down list and uses the following parameters to connect it to a backing bean.

- dataSource="#{ musicBean.listViewType}" – Used to bind the list of types of hierarchical grids getListViewType() method of the MusicDataMB class.
- binding="#{ musicBean.listType}" – Binds the value of the selected item of the drop-down list to the listType member variable of the MusicDataMB class.
- valueChangeListener="#{musicBean.onChange}" – Causes the onChange method of the MusicDataMB class to be called when the value of the drop-down list changes.
- smartRefreshIds="musicGrid" – A comma-delimited list of other JSF components that get updated when the drop-down list is changed or updated. This means that each time the drop-down list is updated, the musicGrid will be updated.

The hierarchical grid is an Infragistics GridView component and is specified using the following two parameters:

- id="musicGrid"
- binding="#{musicBean.hgrid}"

The binding parameter connects the grid to the backing bean and causes the setHgrid() method in the backing bean to get called each time the grid needs to be refreshed.

Related Topic

Configuring the Application - Part 4 of 5 (Section 7.2.4.2.4)

7.2.4.2.4 Configuring the Application - Part 4 of 5

Configuration of this application is simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

For the web.xml file, the following entries need to be added in order to enable the Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <URL-pattern>*.faces</URL-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>server</param-value>
  </context-param>

  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>
        /WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml
    </param-value>
  </context-param>
```

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the MusicDataMB backing bean. To do this, we add the following entry.

In XML:

```
<managed-bean>
  <description>backing bean used by menu grid </description>
  <managed-bean-name>musicBean</managed-bean-name>
  <managed-bean-class>
     com.infragistics.faces.samples.hgrid.MusicDataMB
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 5 of 5 (Section 7.2.4.2.5)

7.2.4.2.5 Deploying the Application - Part 5 of 5

Deploying the application involves putting all the files in the correct location. Below is a brief description of the names and locations of the key files. The Ant build file included in the *Install_Dir*/documentation/tutorial/Dynamic HGrid directory contains tasks to create the WAR file and deploy it to an Apache® Tomcat server. If your environment is different, simply update the build file.

- resources – Contains some of the default images used by the controls.
    - infragistics – Contains the JavaScripts needed by the components.
        - themes – Contains a set of theme subfolders. Each theme subfolder contain all the CSS style sheets and image files needed by the controls. (The default theme is blue.)
- WEB-INF
    - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- hgrid.jsp – The main page that contains the hierarchical grid.
- index.jsp – The home page, which simply redirects the user to the hgrid page through the Faces Servlet.

7.2.4.3 Create a Fixed Column WebGrid

The *WebGrid*™ components are capable of displaying fixed columns. This tutorial will show you how to create a Web page that shows a grid with the first two columns as fixed for scrolling purposes.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the Frozen Columns sample.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.3.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.3.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.4.3.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.4.3.4)**

7.2.4.3.1 Creating the Backing Bean - Part 1 of 4

The Fixed Columns tutorial uses the EmployeeDAO class for its data. This class is used in most of the samples.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the Frozen Columns sample.

The data that will be bound to the grid will be retrieved from a List object. In this tutorial, the data for the list is retrieved from a static array in the class. In a real-world application, the data would most likely come from another data source such as a database.

In Java:

```
/** List of employees to be displayed in the grid*/
private static List employees = null;
```

The employees list is a list of Employee objects. The Employee class is a JavaBean that consists of fields for the first name, last name, e-mail address, and phone number.

To provide the *WebGrid*™ components with access to the employee list, you must provide a getter that follows the JavaBean standard.

In Java:

```
public List getEmployees() {
        return employees;
}
```

That is all the code that is needed for the backing bean. The getEmployees() method will be used by the WebGrid components to display the data as needed.

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.4.3.2)

7.2.4.3.2 Creating the JSP Page - Part 2 of 4

The next step is to create a JSP™ page (dataBind.jsp) that will display a *WebGrid*™ component and bind it to the backing bean that was created in **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.3.1)**.

> **Note:** The full JSP code, source code, and resources needed to build and deploy this tutorial can be found in the Frozen Columns sample.

The following code demonstrates how to create a WebGrid component in the JSP file. See the **Code Explanation** section further below for a description of the code.

In JSP:

```
<ig:gridView
        dataSource="#{webgrid_employeeDAO.employees}"
        pagesize="10"
        fixedColumns="2"
        style="width: 600px;">
        <f:facet name="header">
                <h:outputText value="Employee List" />
        </f:facet>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="First Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.firstName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Last Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.lastName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Email" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.email}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Phone Number" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.phoneNumber}" />
        </ig:column>
</ig:gridView>
```

Code Explanation

The above code is the same as the code in the **Basic Tutorial JSP Page (Section 7.2.4.1.2)** except for the addition of two tags.

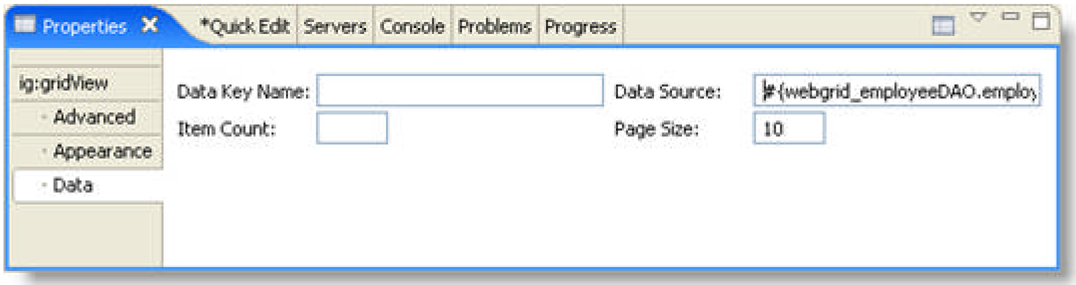
- The fixedColumns tag is used to define the number of columns to be locked for scrolling purposes. The grid starts at the left-most column and locks the number columns you have specified. Since the number is 2 in the code, the first two columns in the grid will be locked for scrolling.

> **Note:** If you want to lock the last two columns in your grid, you need to reorganize your columns so that your last two columns are your first two columns.

- The style tag is used to specify the width for the grid; it is a standard tag. It is required that you set the width, otherwise the scrollbars won't display.

To create a grid at design time using IBM® Rational® Application Developer for WebSphere® 7.0:

1. From the palette, drag the GridView component to your page.
2. In the Properties window, under ig:gridView, select Data .
3. Set the following properties:
   - Data Source: #{webgrid_employeeDAO.employees}
   - Page Size: 10



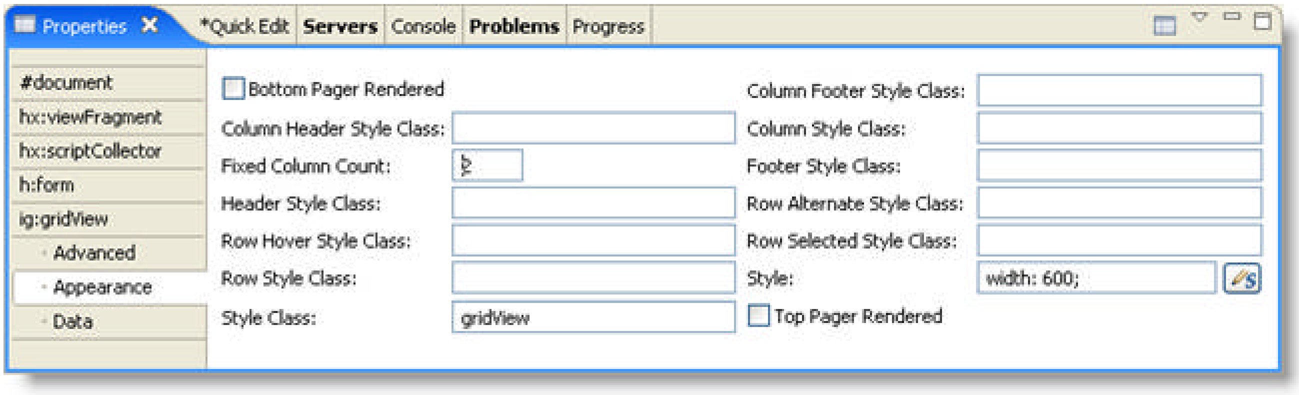
Data Key Name:

Data Source: #{webgrid_employeeDAO.employ

Item Count:

Page Size: 10

4. In the Properties window, under ig:gridView, click Appearance.
5. Set the following properties:
   - Fixed Column Count -- 2
   - Style -- Width: 600



□ Bottom Pager Rendered

Column Header Style Class:

Fixed Column Count: 2

Header Style Class:

Row Hover Style Class:

Row Style Class:

Style Class: gridView

Column Footer Style Class:

Column Style Class:

Footer Style Class:

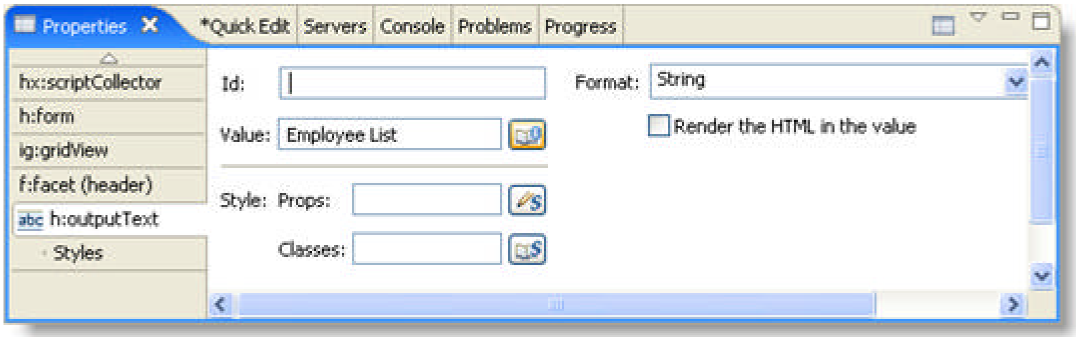
Row Alternate Style Class:

Row Selected Style Class:

Style: width: 600;

□ Top Pager Rendered

6. In the Design view, select the Grid Header and set the following property:
   - Value -- Employee List



Id:

Value: Employee List

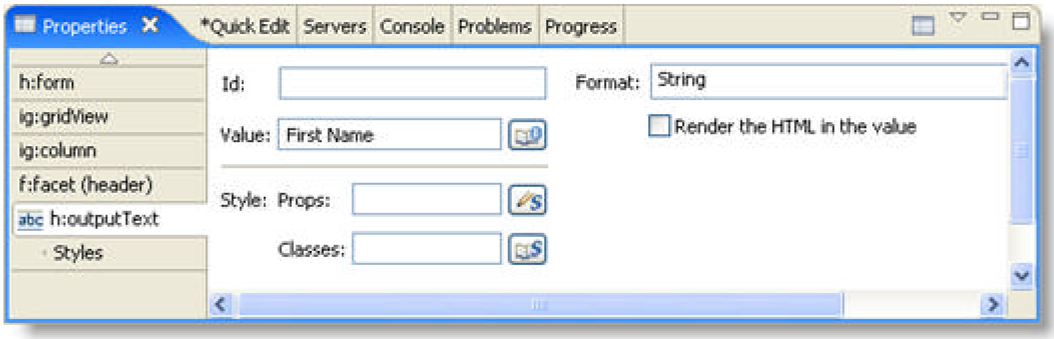
Format: String

□ Render the HTML in the value

Style: Props:

Classes:

7. The default GridView is created with two columns. From the palette, drag the Column component to your grid to add more columns.
8. In the Design view, for each column header, set the Value property:
   - For Column Header 1, set Value to First Name.
   - For Column Header 2, set Value to Last Name.
   - For Column Header 3, set Value to Email.
   - For Column Header 4, set Value to Phone Number.



Id:

Value: First Name

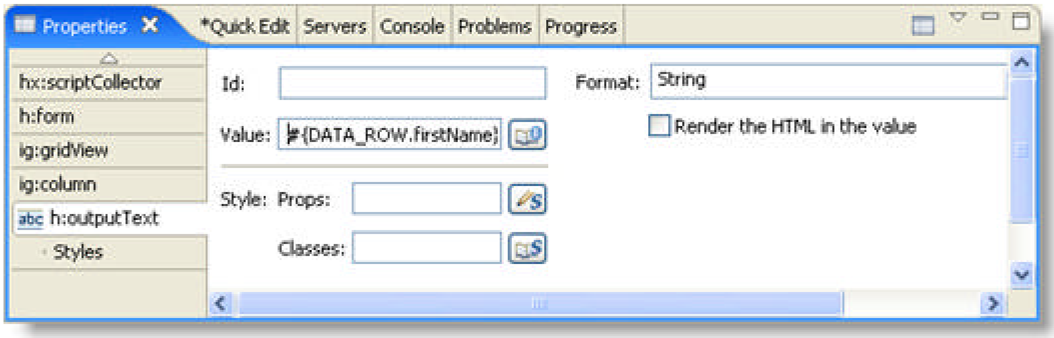
Format: String

□ Render the HTML in the value

Style: Props:

Classes:

9. In the Design view, for each column, set the Value property:
   - For Column 1, set Value to #{DATA_ROW.firstName}.
   - For Column 2, set Value to #{DATA_ROW.lastName}.
   - For Column 3, set Value to #{DATA_ROW.email}.
   - For Column 4, set Value to #{DATA_ROW.phoneNumber}.



Id:

Value: #{DATA_ROW.firstName}

Format: String

□ Render the HTML in the value

Style: Props:

Classes:

Related Topic

Configuring the Application - Part 3 of 4 (Section 7.2.4.3.3)

7.2.4.3.3 Configuring the Application - Part 3 of 4

This topic continues from **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.3.2)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the following:

- web.xml
- managed-beans.xml

The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

web.xml

For the web.xml file, the following entries need to be added to enable Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup> 1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <URL-pattern>*.faces</URL-pattern>
</servlet-mapping>

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
</context-param>

<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
</context-param>
```

managed-beans.xml

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the EmployeeDAO backing bean. To do this, we add the entry below.

In XML:

```
<managed-bean>
  <description>Returns a list of employees</description>
  <managed-bean-name>webgrid_employeeDAO</managed-bean-name>
  <managed-bean-class>
     com.infragistics.faces.samples.data.EmployeeDAO
  </managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.4.3.4)

7.2.4.3.4 Deploying the Application - Part 4 of 4

This topic continues from Updating the **Configuration Files - Part 3 of 4 (Section 7.2.4.3.3)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
  - /lib -- Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
  - web.xml
  - managed-beans.xml
  - navigation.xml
  - faces-config.xml
- dataBind.jsp -- The main page that contains the searchable employee grid.
- index.jsp -- The home page which simply redirects the user to the dataBind page through the Faces Servlet.

7.2.4.4 Create a Hierarchical WebGrid

The *WebGrid*™ components can show related data in hierarchical structure. This tutorial will show you how to create a Web page that shows a grid displaying data hierarchically.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the Hierarchical Grid Sample.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.4.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.4.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.4.4.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.4.4.4)**

7.2.4.4.1 Creating the Backing Bean - Part 1 of 4

The first thing that needs to be done is to create the backing bean that will be used to model the data and actions required by the Web page. The Web page contains only one component -- the WebGrid component, which displays hierarchical data.

To model this component, we will create three Java classes called Index, BaseGridPage, Company, and DAO.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the Hierarchical Grid Sample.

Index.java

The Index class extends the BaseGridPage.

In Java:

```
import com.infragistics.faces.samples.webgrid.BaseGridPage;

public class Index extends BaseGridPage {

}
```

BaseGridPage.java

The BaseGridPage class contains the getters that are called to fetch the data out of the DAO.java file.

In Java:

```
import com.infragistics.faces.samples.data.DAO;

public class BaseGridPage {
        public List getCompanies() {
                return DAO.getCompanies();
        }

        public List getEmployees() {
                return DAO.getEmployees();
        }

        public void sortListener(SortEvent evnt) {
                System.out.println("A SortEvent has been fired");
        }
}
```

DAO.java

The DAO class contains the data that is being shown in the grid, and contains the calls to the Company.java file that creates all the company objects and related employee objects. This class also keeps a static member of type List that contains all the company objects once they are created.

In Java:

```
private static List cachedCompanies = null;
```

The getCompanies method is called by the BaseGridPage class, and checks to see if the companies have been loaded into the static list. If it hasn't been loaded, it makes a call to the buildCompanies method.

In Java:

```
public static List getCompanies() {
        if (cachedCompanies==null) {
                cachedCompanies=buildCompanies();
        }

        return cachedCompanies;
}
```

The buildCompanies method creates the company objects, and returns them to the getCompanies method.

In Java:

```
private static List buildCompanies() {
        List result = new ArrayList();
        for (int i=0; i<companies.length; i++) {
                Company company =
                        new Company(
                                i+1,
                                companies[i][0],
                                companies[i][1],
                                companies[i][2]
                        );
                result.add(company);
        }
        return result;
}
```

Company.java

The Company class contains the definition of a company. The key code in this class is how it makes calls to create employees, and how it relates those employees to each company. To create that relation, each company has a list of employees.

In Java:

```
private List employees;
```

The employees list is populated by the getEmployees method in the Company class. This method makes a call to the initEmployees method, which itself calls the getEmployees method from the DAO class.

In Java:

```java
public List getEmployees() {
        if (employees==null)
                employees = initEmployees();
        return employees;
}

private List initEmployees() {
        List result = new ArrayList();
        if (employees==null) {
                Iterator tmp = DAO.getEmployees().iterator();
                while (tmp.hasNext()) {
                        Person aPerson = (Person) tmp.next();
                        if ((this.getId()%10)==(aPerson.getId()%10)) {
                                result.add(aPerson);
                        }
                }
        }
        return result;
}
```

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.4.4.2)

7.2.4.4.2 Creating the JSP Page - Part 2 of 4

This topic continues from **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.4.1)**.

The next step is to create a JSP™ page (dataBind.jsp) that will display the components and bind the values and actions to the backing bean.

> **Note:** The full JSP code, source code and resources needed to build and deploy this tutorial can be found in the Hierarchical Grid Sample.

To display the hierarchical company and employee data, NetAdvantage for JSF's *WebGrid*™ will be used. As shown in the JSP code below, the WebGrid components have two bindings:

- dataSource= "#{webgrid_hierarchicalGridPage.companies}" -- This binding binds the companies list of the backing bean to the WebGrid components.
- dataSource= "#{DATA_ROW.employees.}" -- This binding binds the employees list of the backing bean to another set of WebGrid components, shown when the expansion indicator is selected.

In JSP:

```
<ig:gridView
    dataSource="#{webgrid_hierarchicalGridPage.companies}"
    pagesize="5">
    <f:facet name="header">
        <h:outputText value="Company List" />
    </f:facet>
    <ig:column>
        <f:facet name="masterDetail">
            <ig:gridView
                dataSource="#{DATA_ROW.employees}"
                pagesize="4">
                <f:facet name="header">
                    <h:outputText value="Employees List" />
                </f:facet>
                <ig:column sortBy="firstName">
                    <f:facet name="header">
                        <h:outputText value="First Name" />
                    </f:facet>
                    <h:outputText value="#{DATA_ROW.firstName}" />
                </ig:column>
                <ig:column>
                    <f:facet name="header">
                        <h:outputText value="Last Name" />
                    </f:facet>
                    <h:outputText value="#{DATA_ROW.lastName}" />
                </ig:column>
                <ig:column>
                    <f:facet name="header">
                        <h:outputText value="Email" />
                    </f:facet>
                    <h:outputText value="#{DATA_ROW.email}" />
                </ig:column>
                <ig:column>
                    <f:facet name="header">
                        <h:outputText value="Phone Number" />
                    </f:facet>
                    <h:outputText value="#{DATA_ROW.phoneNumber}" />
                </ig:column>
            </ig:gridView>
        </f:facet>
    </ig:column>
    <ig:column>
        <f:facet name="header">
            <h:outputText value="Company Name" />
        </f:facet>
        <h:outputText value="#{DATA_ROW.name}" />
    </ig:column>
    <ig:column>
        <f:facet name="header">
            <h:outputText value="City" />
        </f:facet>
        <h:outputText value="#{DATA_ROW.city}" />
    </ig:column>
    <ig:column>
        <f:facet name="header">
            <h:outputText value="Country" />
        </f:facet>
        <h:outputText value="#{DATA_ROW.country}" />
    </ig:column>
</ig:gridView>
```

To create the hierarchical grid, do the following:

1. Create the initial or parent grid, and bind it to a data source. Then create all of its columns.
2. In the parent grid, directly after setting the header, create a column tag, which has a facet tag with its Name attribute set to "masterDetail".

   > **Note:** The location of the column containing the facet tag controls where the expansion indicator column will appear in your grid. For example, if you place this column as the third column in your grid, the expansion indicator will be located in the third column when the grid renders.

3. In between the facet tags created in step 2, create a WebGrid, and set its dataSource property to DATA_ROW.employees, and create all of its columns.

Related Topic

Configuring the Application - Part 3 of 4 (Section 7.2.4.4.3)

7.2.4.4.3 Configuring the Application - Part 3 of 4

This topic continues from **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.4.2)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the following:

- web.xml
- managed-beans.xml

The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

web.xml

For the web.xml file, the following entries need to be added to enable Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup> 1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <URL-pattern>*.faces</URL-pattern>
</servlet-mapping>

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
</context-param>

<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>
      /WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml
  </param-value>
</context-param>
```

managed-beans.xml

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the Index backing bean. To do this, we add the entry below.

In XML:

```
<managed-bean>
  <description>backing bean for the page hierarchicalGrid</description>
  <managed-bean-name>webgrid_hierarchicalGridPage</managed-bean-name>
  <managed-bean-class>
       com.infragistics.faces.samples.webgrid.hierarchicalGrid.Index
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.4.4.4)

7.2.4.4.4 Deploying the Application - Part 4 of 4

This topic continues from **Configuring the Application - Part 3 of 4 (Section 7.2.4.4.3)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
    - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the Index, Company, DAO, and BaseGridPage classes.
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- dataBind.jsp – The main page that contains the hierarchical grid.
- index.jsp – The home page which simply redirects the user to the dataBind page through the Faces Servlet.

7.2.4.5 Dynamically Create a Fixed Column WebGrid

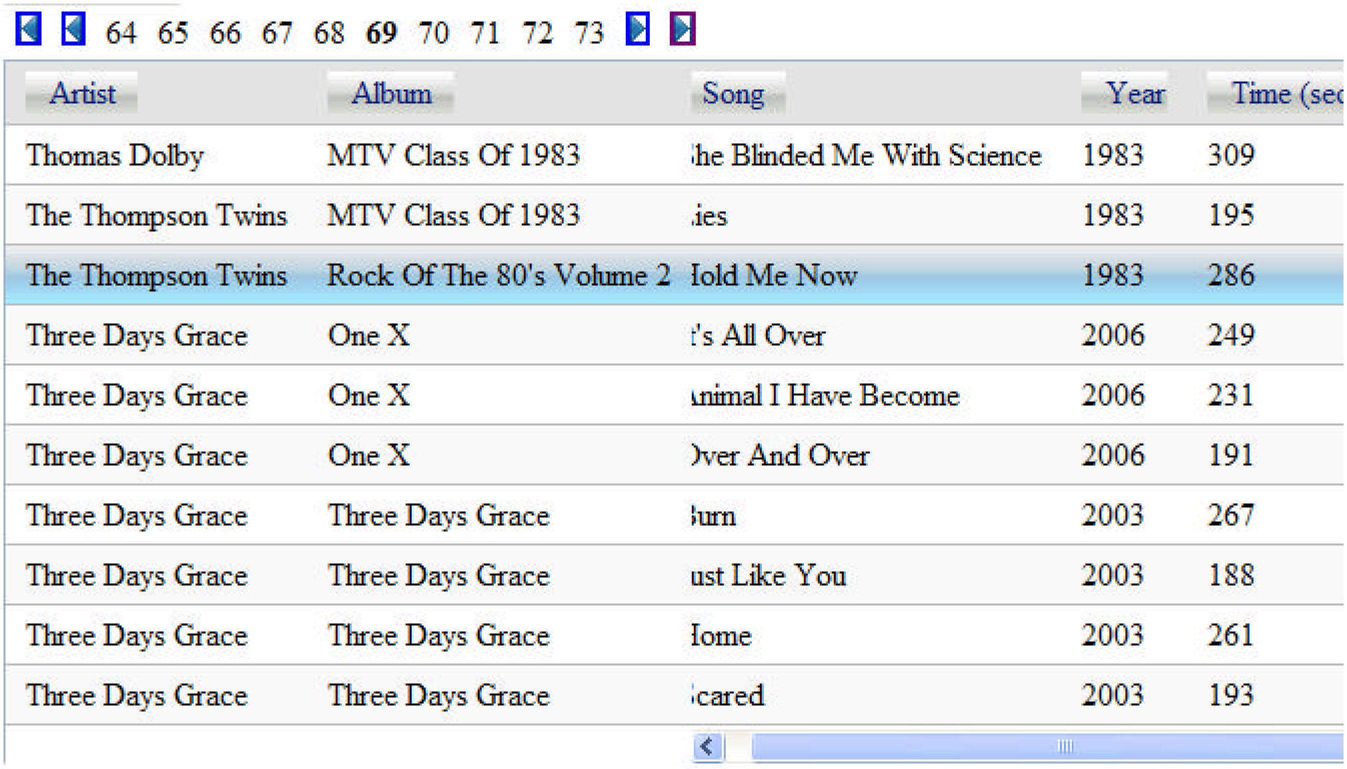
Creating a set of fixed columns in a grid is just a matter of setting the fixedColumnCount attribute on the grid in the JSP page. Creating fixed columns this way is fairly static. Although a user can make an unfixed column fixed by dragging it to the fixed area of the grid this can be difficult if the grid is wide. This tutorial will show how to dynamically create a fixed column grid that will have a menu in the column header that will allow the user to easily fix or unfix a column.

For this tutorial we are going to create a grid to display information about a list of songs. There will be columns for Artist, Album, Song, Year, Length of Song, Track Num, and Date Added. On the top of each column will be a menu that the user can select to make the column fixed or unfixed. If a user makes a fixed column unfixed it will be moved to the left most column of the unfixed area of the grid. If the user make an unfixed column fixed it will be moved to the right most column of the fixed area of the grid.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documents/tutorial/FixedColumns directory.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.5.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.5.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.4.5.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.4.5.4)**



64 65 66 67 68 **69** 70 71 72 73

Artist	Album	Song	Year	Time (sec
Thomas Dolby	MTV Class Of 1983	;he Blinded Me With Science	1983	309
The Thompson Twins	MTV Class Of 1983	.ies	1983	195
The Thompson Twins	Rock Of The 80's Volume 2	Iold Me Now	1983	286
Three Days Grace	One X	I's All Over	2006	249
Three Days Grace	One X	ınimal I Have Become	2006	231
Three Days Grace	One X	)ver And Over	2006	191
Three Days Grace	Three Days Grace	;urn	2003	267
Three Days Grace	Three Days Grace	ust Like You	2003	188
Three Days Grace	Three Days Grace	Iome	2003	261
Three Days Grace	Three Days Grace	;cared	2003	193

7.2.4.5.1 Creating the Backing Bean - Part 1 of 4

The backing bean for this tutorial will use to dynamically create the fixed column grid. The Web page will contain a grid that has menus in the column headers

The grid is created dynamically by the buildGrid() method. This method is called anytime the setGrid() method is called. This method dynamically builds the grid column by column based on data in two column lists m_fixedColumnList and m_unfixedColumnList. This list contains the data needed to build the fixed and unfixed columns of the grid respectively. The data source bound to the list is just a set of list of MusicData. For a brief description of the MusicData structure see part 1 of the Dynamic Hierarchical grid tutorial.

In Java:

```
private GridView buildGrid(GridView grid)
{
  Boolean isInitialised = null;
  Object attr = grid.getAttributes().get(IS_INITIALIZED);
  if (attr != null)
  {
    isInitialised = (Boolean) (attr);
  }
  else
  {
    isInitialised = new Boolean(false);
  }
  if (!isInitialised)
  {
    clearGrid();
    // create the data for the grid
    createMusicData();

    // get a reference to the JSF Application,
    // we need it to create value bindings later-on
    Application application = FacesContext.getCurrentInstance().getApplication();
    UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();

    // bind the grid to display a list of data
    grid.setValueBinding("dataSource", application
          .createValueBinding("#{fixedColumn.musicData}"));

    // set style with width of the grid if this is not set
    // the scroll bar for the unfixed column will not show
    ValueBinding vb = application.createValueBinding("#{fixedColumn.gridStyle}");
    grid.setValueBinding("style", vb);

    // display 10 items at a time
    grid.setPageSize(10);

    //Add fixed columns to the grid
    for(int i = 0; i < m_fixedColumnList.size(); i++)
    {
      ColumnData coldata = (ColumnData)m_fixedColumnList.get(i);
      String vbExpression = "#{DATA_ROW." + colData.getGetter() + "}";
      Column aColumn = createColumn(colData.getName(), vbExpression, application,
             viewRoot);
      aColumn.setSortBy(colData.getGetter());
      grid.getTemplateItems().add(aColumn);
    }
    //Set number of fixed columns
    grid.setFixedColumnCount(m_fixedColumnList.size());
    // Add unfixed columns to the grid
    for(int i = 0; i < m_unfixedColumnList.size(); i++)
    {
      ColumnData coldata = (ColumnData)m_unfixedColumnList.get(i);
      String vbExpression = "#{DATA_ROW." + colData.getGetter() + "}";
      Column aColumn = createColumn(colData.getName(), vbExpression, application,
             viewRoot);
      aColumn.setSortBy(colData.getGetter());
      grid.getTemplateItems().add(aColumn);
    }

    grid.getAttributes().put(IS_INITIALIZED,
    new Boolean(true));
  }
  return (grid);
}
```

The buildGrid() method uses the createColumn() method to create each column object. The createColumn() method in turn calls the createHeaderMenu() method to create the menu for each header. This method dynamically builds the menu for each column header based on the column name. The method creates a method binding for each menu item “Fix Column” and “UnFix Column”. When the user selects these menu items the method associated with them is called.

In Java:

```
private HTMLMenu createHeaderMenu(String colName, Application application,
  UIViewRoot viewRoot)
{
  MethodBinding fixedActionListener = application.createMethodBinding(
              "#{fixedColumn.setColumnFixed}",
              new Class[] { javax.faces.event.ActionEvent.class });

  MethodBinding unfixedActionListener = application.createMethodBinding(
          "#{fixedColumn.setColumnUnfixed}",
          new Class[] { javax.faces.event.ActionEvent.class });

  // Create the menu for the column header
  HTMLMenu menu = (HTMLMenu) application.createComponent(HTMLMenu.COMPONENT_TYPE);

  //Create menu item using column name
  MenuItem menuItem = (MenuItem)
                      application.createComponent(MenuItem.COMPONENT_TYPE);
  menuItem.setValue(colName);

  //Create menu sub items for the two menu options
  MenuItem menuSubItem = (MenuItem)
                      application.createComponent(MenuItem.COMPONENT_TYPE);

  //menuSubItem.setId(viewRoot.createUniqueId());
  menuSubItem.setValue("Fix Column");
  menuItem.getChildren().add(menuSubItem);
  //set actionlistener to respond to menu event
  menuSubItem.setActionListener(fixedActionListener);

  menuSubItem = (MenuItem) application.createComponent(MenuItem.COMPONENT_TYPE);

  //menuSubItem.setId(viewRoot.createUniqueId());
  menuSubItem.setValue("UnFix Column");
  menuItem.getChildren().add(menuSubItem);
  menuSubItem.setActionListener(unfixedActionListener);

  //add the menu item to the menu
  menu.getChildren().add(menuItem);

  return(menu);
}
```

The "Fix Column" menu item calls the setColumnFixed() method and the "UnFix Column" menu item calls the setColumnUnfixed() method. All these methods do is to move the column data between the m_fixedColumnList and m_unfixedColumnList based whether the column is being fixed or unfixed and then rebuilds the grid.

In Java:

```
/**
* Sets the selected column to be unfixed in the grid
* by moving the column data of the selected column
* to the unfixed list.
*
* @param event
*/
public void setColumnUnfixed(ActionEvent event)
{
        MenuItem selectedMenu = (MenuItem)event.getSource();
        MenuItem colMenu = (MenuItem)selectedMenu.getParent();
        String colName = colMenu.getValue().toString();
        moveToUnfixedList(colName);
        System.out.println("UnFixing Column " + colName);
        //rebuild to grid to reset the columns
        m_grid.getAttributes().put(IS_INITIALIZED,
          new Boolean(false));
        buildGrid(m_grid);
}
```

For more details on the methods of the backing bean take a look at the source code for this tutorial.

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.4.5.2)

7.2.4.5.2 Creating the JSP Page - Part 2 of 4

The next step is to create a JSP page (fixedColumns.jsp) that will display the components and bind the values and actions to the backing bean. (Note: The full JSP code, source code and resources needed to build and deploy this tutorial can be found in the Install_Dir/documents/tutorial/FixedColumns directory.)

Besides the standard HTML and tags the JSP contains the tags the GridView component.

In JSP:

```
<f:view>
   <h:form>
      <h:panelGroup>
         <h:panelGroup binding="#{fixedColumn.topPanel}">
      </h:panelGroup>
      <h:panelGroup>
         <ig:gridView id="musicGrid"
                      binding="#{fixedColumn.grid}" >
         </ig:gridView>
      </h:panelGroup>
   </h:panelGroup>
   </h:form>
</f:view>
```

The grid is an Infragistics GridView component is and is specified using to parameters id=”musicGrid” and binding="#{fixedColumn.grid}" The binding parameter connects the grid to the backing bean and will cause the setGrid() method in the backing bean to get called anytime the grid needs to be refreshed.

Related Topic

Configuring the Application - Part 3 of 4 (Section 7.2.4.5.3)

7.2.4.5.3 Configuring the Application - Part 3 of 4

Configuration of this application is simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed. For the web.xml file, the following entries need to be added to enable faces servlet and define some JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <URL-pattern>*.faces</URL-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>server</param-value>
  </context-param>

  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
  </context-param>
```

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the FixedColumnMB backing bean. To do this we add the following entry.

In XML:

```
<managed-bean>
  <description>backing bean used by menu grid </description>
  <managed-bean-name>fixedColumn</managed-bean-name>
  <managed-bean-class>
       com.infragistics.faces.samples.fixedColumn.FixedColumnMB
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.4.5.4)

7.2.4.5.4 Deploying the Application - Part 4 of 4

Deploying the application is just a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files. The ant build file included in the *Install_Dir*/documents/tutorial/Dynamic HGrid directory contains tasks to create the WAR file and deploy it to a Tomcat server. If your environment is different, simply update the build file.

- resources – Contains some of the default images used by the controls
    - infragistics – Contains the JavaScripts needed by the components.
        - themes – Contains a set of theme sub-folders. Each theme sub-folder contain all the CSS style sheets and image files needed by the controls. (The default theme is blue.)
- WEB-INF
    - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- fixedColumn.jsp – Main page that contains the hierarchical grid.
- index.jsp – Home page which simply redirects the user to the fixedColumn page through the FacesServlet.

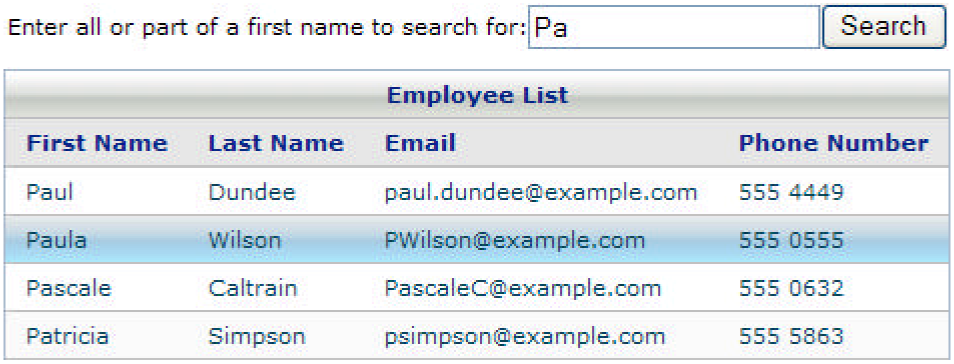
7.2.4.6 Dynamically Update Data in a WebGrid

The *WebGrid*™ components have the ability to dynamically update the data displayed based on changes in the data bound to WebGrid. This tutorial will show you how to create a Web page that filters a set of data in a WebGrid component based on text entered in a text box. The Web page that will be created will contain a grid that displays a list of employee data (first name, last name, e-mail address, and phone number). The Web page will also contain a text box and a Search button. If the end user enters all or part of an employee's first name, and then clicks the Search button, the data in WebGrid will be filtered based on the first name entered.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/GridSearch directory.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.6.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.6.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.4.6.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.4.6.4)**



Enter all or part of a first name to search for: Pa

Employee List

First Name	Last Name	Email	Phone Number
Paul	Dundee	paul.dundee@example.com	555 4449
Paula	Wilson	PWilson@example.com	555 0555
Pascale	Caltrain	PascaleC@example.com	555 0632
Patricia	Simpson	psimpson@example.com	555 5863

7.2.4.6.1 Creating the Backing Bean - Part 1 of 4

The first thing that needs to be done is to create the backing bean that will be used to model the data and actions required by the Web page. The Web page will contain three main components:

- the WebGrid component to display employee data
- a text box to input the employee’s first name
- a search button that will be used to initiate the search

To model these components, we will create a Java class called EmployeeDAO.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/GridSearch directory.

A member variable will be used to model the text box and WebGrid as follows:

In Java:

```
/** Handle to the first name search input field*/
private UIInput m_searchStr = null;
/** Handle to the grid component on the page*/
private GridView m_employeeGridView = null;
```

The text box is modeled using the standard JSF UIInput class and is called m_searchStr. WebGrid is modeled using NetAdvantage for JSF's gridView component and is called m_employeeGridView. In order to provide the Web page with access to these objects, you must provide setters and getters that follow the JavaBean standard.

In Java:

```
public GridView getEmployeeGridView() {
        return m_employeeGridView;
}

public void setEmployeeGridView(GridView employeeGridView) {
        m_employeeGridView = employeeGridView;
}
public UIInput getSearchStr(){
        return(m_searchStr);
}
public void setSearchStr(UIInput searchStr){
        m_searchStr = searchStr;
}
```

The data that will be bound to the grid will be retrieved from a List object. In this tutorial, for simplicity purposes, the data for the list is retrieved from a static array in the class. If this had been a real-world application, the data would most likely have come from another data source such as a database.

In Java:

```
/** List of employees that will be display in the grid*/
private static List employees = null;
```

The Search button is not modeled in the EmployeeDAO class because we do not need to access its value or properties. The only thing that has to be modeled for the Search button is the action that will take place when the button is clicked. To do this, we create an applySettings method that will be called when the Search button is clicked.

In Java:

```
/**
* Action method called when the Search button is pressed
* This method gets the search string from the input field
* and then filters the employee list by first name
* using the search string
*
* @param event
*/
public void applySettings(ActionEvent event){
        //Reset list to full value
        buildEmpList();
        //Get value of first name search string
        String searchStr = m_searchStr.getValue().toString();
        if(searchStr.length() > 0)
        {
                filterEmpList(searchStr);
        }
        m_employeeGridView.dataBind();
}
```

This method first resets the employees list so that it contains all possible employees. The next step that occurs is the search string is retrieved from the text box. If the searchStr is empty, then nothing is done; otherwise the filterEmpList() method is called. The filterEmpList() method creates a new employees list that contains only the employees whose first name matches the search criteria. After the filterEmpList() method is complete, the dataBind() method of m_employeeGridView is called. This causes WebGrid to get the filtered version of the employees list and update its display.

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.4.6.2)

7.2.4.6.2 Creating the JSP Page - Part 2 of 4

This topic continues from **Creating the Backing Bean - Part 1 of 4 (Section 7.2.4.6.1)**.

The next step is to create a JSP™ page (dataBind.jsp) that will display the components and bind the values and actions to the backing bean.

> **Note:** The full JSP code, source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/GridSearch directory.

A standard JSF inputText tag is used to enter the search string as show here.

In JSP:

```
<h:inputText value=""
        binding="#{webgrid_employeeDAO.searchStr}"/>
```

The inputText component contains a binding expression to bind this component to the searchStr variable of the EmployeeDAO backing bean whose alias is “webgrid_employeeDAO”, as defined in the managed-beans.xml file.

The Search button is represented by the standard JSF commandButton. As shown below, the commandButton is connected to the applySettings method of the backing bean through the actionListener tag.

In JSP:

```
<h:commandButton
        actionListener="#{webgrid_employeeDAO.applySettings}"
        value="Search">
</h:commandbutton>
```

To display the employee data, NetAdvantage for JSF’s *WebGrid™* will be used. As shown in the JSP code below, WebGrid has two bindings. The first binding (binding = "#{webgrid_employeeDAO.employeeGridView}") provides the backing bean with access to the WebGrid so that it can be updated when the employee data changes. The second binding (dataSource= "#{webgrid_employeeDAO.employees}") binds the employees list of the backing bean to WebGrid.

In JSP:

```
<ig:gridView binding="#{webgrid_employeeDAO.employeeGridView}"
             dataSource="#{webgrid_employeeDAO.employees}"
             pageSize="10">
   <f:facet name="header">
      <h:outputText value="Employee List" />
   </f:facet>
   <ig:column sortBy="firstName">
      <f:facet name="header">
         <h:outputText value="First Name" />
      </f:facet>
      <h:outputText value="#{DATA_ROW.firstName}" />
   </ig:column>
   <ig:column sortBy="lastName">
      <f:facet name="header">
         <h:outputText value="Last Name" />
      </f:facet>
      <h:outputText value="#{DATA_ROW.lastName}" />
   </ig:column>
   <ig:column sortBy="email">
      <f:facet name="header">
         <h:outputText value="Email" />
      </f:facet>
      <h:outputText value="#{DATA_ROW.email}" />
   </ig:column>
   <ig:column sortBy="phoneNumber">
      <f:facet name="header">
         <h:outputText value="Phone Number" />
      </f:facet>
      <h:outputText value="#{DATA_ROW.phoneNumber}"/>
   </ig:column>
</ig:gridView>
```

Related Topic

Configuring the Application - Part 3 of 4 (Section 7.2.4.6.3)

7.2.4.6.3 Configuring the Application - Part 3 of 4

This topic continues from **Creating the JSP Page - Part 2 of 4 (Section 7.2.4.6.2)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the following:

- web.xml
- managed-beans.xml

The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

web.xml

For the web.xml file, the following entries need to be added to enable Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup> 1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <URL-pattern>*.faces</URL-pattern>
</servlet-mapping>

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
</context-param>

<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
</context-param>
```

managed-beans.xml

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the EmployeeDAO backing bean. To do this, we add the entry below.

In XML:

```
<managed-bean>
  <description>Returns a list of employees</description>
  <managed-bean-name>webgrid_employeeDAO</managed-bean-name>
  <managed-bean-class>
       com.infragistics.faces.samples.data.EmployeeDAO
  </managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
```

```
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.4.6.4)

7.2.4.6.4 Deploying the Application - Part 4 of 4

This topic continues from **Configuring the Application - Part 3 of 4 (Section 7.2.4.6.3)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files. The Apache® Ant build file included in the *Install_Dir*/documentation/tutorials/GridSearch directory contains tasks to create the WAR file and deploy it to an Apache Tomcat server. If your environment is different, you must update the build file.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
  - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
  - web.xml
  - managed-beans.xml
  - navigation.xml
  - faces-config.xml
- dataBind.jsp – The main page that contains the searchable employee grid.
- index.jsp – The home page which simply redirects the user to the dataBind page through the Faces Servlet.

7.2.4.7 Working with Facelets

The NetAdvantage for JSF® components have the ability to run in Facelets. This tutorial will show you how to create a sortable grid of employee data that will be displayed on a Facelet page.

Using JSF components in a Facelet application is very similar using them on a JSF page. The main difference with an application that uses Facelets is the way you create the individual pages. Facelets allows the creation of templates, which provide a basic layout and format for a page. Each individual page then references a template and fills in the required pieces to complete the page.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the **Knowledge Base 10003 (http://devcenter.infragistics.com/Support/KnowledgeBaseArticle.aspx?ArticleID=10003)** directory.

- **Creating the Backing Bean - Part 1 of 5 (Section 7.2.4.7.1)**
- **Creating the Template - Part 2 of 5 (Section 7.2.4.7.2)**
- **Creating the JSP Page - Part 3 of 5 (Section 7.2.4.7.3)**
- **Configuring the Application - Part 4 of 5 (Section 7.2.4.7.4)**
- **Deploying the Application - Part 5 of 5 (Section 7.2.4.7.5)**

7.2.4.7.1 Creating the Backing Bean - Part 1 of 5

To model the list of employees, we will create a Java class called EmployeeDAO.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the **Knowledge Base 10003 (http://devcenter.infragistics.com/Support/KnowledgeBaseArticle.aspx?ArticleID=10003)**

The data that will be bound to the grid will be retrieved from a List object. In this tutorial, for simplicity purposes, the data for the list is retrieved from a static array in the class. In a real-world application, the data would most likely come from another data source such as a database.

In Java:

```
/** List of employees to be displayed in the grid*/
private static List employees = null;
```

The employees list is simply a list of Employee objects. The Employee class is a JavaBean that consists of fields for the first name, last name, e-mail address, and phone number.

To provide the WebGrid™ components with access to the employee list, you must provide a getter that follows the JavaBean standard.

In Java:

```
public List getEmployees() {
        return employees;
}
```

That is all the code that is needed for the backing bean. The getEmployees() method will be used by the WebGrid to display the data as needed.

Related Topic

Creating the JSP Page - Part 3 of 5 (Section 7.2.4.7.3)

7.2.4.7.2 Creating the Template - Part 2 of 5

The next step is to create a template file (common.xHTML) that contains the basic format and style that we want to use for the page. The template contains a parameter for the page header and page body that needs to be specified by each page the uses this template. The full code of this template is shown below.

Note: The template contains the <h:form> tags inside the body tags. This is required for the AJAX functionality of our NetAdvantage for JSF to work.

Note: The full source code and resources needed to build and deploy this tutorial can be found in the **Knowledge Base 10003 (http://devcenter.infragistics.com/Support/KnowledgeBaseArticle.aspx?ArticleID=10003)**

In JSP:

```
<body bgcolor="#ffffff">
  <h:form>
  <table style="border:1px solid #CAD6E0"  align="center"
                   cellpadding="0" cellspacing="0" border="0" width="400">
    <tbody>
      <tr>
        <td class="header" height="42" align="center" valign="middle"
                      width="100%" bgcolor="#E4EBEB">
          <ui:insert name="pageHeader">Page Header</ui:insert>
        </td>
      </tr>
      <tr>
        <td height="1" width="100%" bgcolor="#CAD6E0"></td>
      </tr>
      <tr>
        <td width="100%"  colspan="2">
          <table width="100%" style="height:150px" align="left"
                    cellpadding="0" cellspacing="0" border="0">
            <tbody>
              <tr>
                <td align="center" width="100%" valign="middle">
                  <ui:insert name="body">Page Body</ui:insert>
                </td>
              </tr>
            </tbody>
          </table>
        </td>
      </tr>
      <tr>
        <td colspan="2" valign="bottom" height="1" width="100%"
                       bgcolor="#CAD6E0"></td>
      </tr>
    </tbody>
  </table>
</h:form>
</body>
```

Code Explanation

The above code defines the template, which outlines the basic format and style that we want to use for the page. The following list the key sections defined in the template code.

1. The template contains a parameter for the page header and page body that needs to be specified by each page that uses this template.
    - pageHeader insert -- inserts the information that you specify in the tags in your facelet page.
    - body insert -- inserts the information you specify in your body tags in your facelet page.

7.2.4.7.3 Creating the JSP Page - Part 3 of 5

The next step is to create a facelet page (employees.xHTML) that will display a WebGrid component and bind it to the backing bean that was created in **Creating the Backing Bean - Part 1 of 5 (Section 7.2.4.7.1)**. We also reference the template file called common.xHTML that was created in **Creating the Template - Part 2 of 5 (Section 7.2.4.7.2)**. See the **Code Explanation** section further below for a description of the code.

> **Note:** The full JSP code, source code, and resources needed to build and deploy this tutorial can be found in the **Knowledge Base 10003 (http://devcenter.infragistics.com/Support/KnowledgeBaseArticle.aspx?ArticleID=10003#Code%20Explanation)**

In JSP:

```
<HTML xmlns="http://www.w3.org/1999/xHTML"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/HTML"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:c="http://java.sun.com/jstl/core"
    xmlns:fmt="http://java.sun.com/jstl/fmt"
    xmlns:acegijsf="http://sourceforge.net/projects/jsf-comp/acegijsf"
    xmlns:t="http://myfaces.apache.org/tomahawk"
    xmlns:ig="http://www.infragistics.com/faces/netadvantage">

<ui:composition template="/templates/common.xHTML">
    <ui:define name="pageTitle">Infragistics Grid Test</ui:define>
    <ui:define name="pageHeader">Employee List</ui:define>
        <ui:define name="body">
            <ig:gridView
                dataSource="#{employeeDAO.employees}"
                pagesize="10">
                <f:facet name="header">
                    <h:outputText value="Employees:" />
                </f:facet>
                <ig:column sortBy="firstName">
                <f:facet name="header">
                    <h:outputText value="First Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.firstName}" />
            </ig:column>
            <ig:column sortBy="lastName">
                <f:facet name="header">
                    <h:outputText value="Last Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.lastName}" />
            </ig:column>
            <ig:column sortBy="email">
                <f:facet name="header">
                    <h:outputText value="Email" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.email}" />
            </ig:column>
            <ig:column sortBy="phoneNumber">
                <f:facet name="header">
                    <h:outputText value="Phone Number" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.phoneNumber}" />
            </ig:column>
        </ig:gridView>
    </ui:define>
</ui:composition>
</HTML>
```

Code Explanation

The following is a list of the steps that occur in the above code.

1. The tag ig:gridView creates the WebGrid component. This tag has the following two properties:
   - dataSource= "#{webgrid_employeeDAO.employees} -- This property binds the Employees list of the backing bean to the WebGrid component.
   - pagesize="10" -- This property indicates to the WebGrid component to display 10 rows per page.
2. The next block of code specifies a header area in the WebGrid component where you can place a title for the data being shown.
3. The next block of code is a series of ig:column tags that are used to define the column header and data value for each column of the WebGrid component. For this example, the data value for each column is a simple outputText component that is bound to one of the fields of the Employee object. The DATA_ROW part of the value binding is a built-in constant that is used to enumerate the rows of the grid when it is rendered. See **Identify the Rows in a Collection (Section 4.7.3.1.5)** for more information.

Note: The WebGrid component is simply a container so the data value for a column can be represented by any type of component, including drop-down lists, check boxes, inputText fields, etc.

Related Topic

Configuring the Application - Part 4 of 5 (Section 7.2.4.7.4)

7.2.4.7.4 Configuring the Application - Part 4 of 5

This topic continues from **Creating the JSP Page - Part 3 of 5 (Section 7.2.4.7.3)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the following:

- web.xml
- managed-beans.xml

The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed.

web.xml

For the web.xml file, the following entries need to be added to enable Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <URL-pattern>*.faces</URL-pattern>
</servlet-mapping>

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
</context-param>

<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
</context-param>

<context-param>
  <param-name>facelets.LIBRARIES</param-name>
  <param-value>
      /WEB-INF/taglib/netAdvantage.facelet.taglib.xml
  </param-value>
</context-param>
```

managed-beans.xml

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the EmployeeDAO backing bean. To do this, we add the entry below.

In XML:

```
<managed-bean>
  <description>Returns a list of employees</description>
  <managed-bean-name>webgrid_employeeDAO</managed-bean-name>
  <managed-bean-class>
       com.infragistics.faces.samples.data.EmployeeDAO
  </managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 5 of 5 (Section 7.2.4.7.5)

7.2.4.7.5 Deploying the Application - Part 5 of 5

This topic continues from **Configuring the Application - Part 4 of 5 (Section 7.2.4.7.4)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files.

- /resources -- Contains some of the default images used by the components.
- /resources/infragistics -- Contains the JavaScript needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
  - lib – Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class.
  - web.xml
  - managed-beans.xml
  - navigation.xml
  - faces-config.xml
- dataBind.jsp – The main page that contains the searchable employee grid.
- index.jsp – The home page which simply redirects the user to the dataBind page through the Faces Servlet.

7.2.5 WebInput

In this section, you’ll find a step-by-step procedure that walks you through a scenario that is common when using the *WebInput* control.

- **Using Smart Refresh (Section 7.2.5.1)**

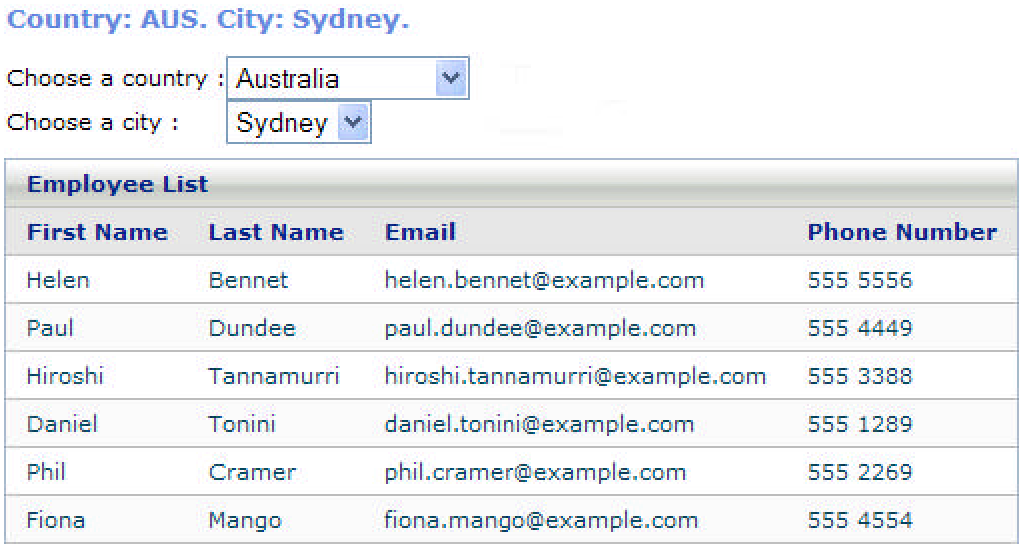
7.2.5.1 Using Smart Refresh

The *WebInput*™ components that are used for selection (i.e., CheckBox, CheckBoxList, DropDownList, RadioButton, and RadioButtonList) have the ability to cause other components to be updated when their values change. This tutorial will show you how to create a Web page that will update an output text field and a grid when a value is selected from a drop-down list.

For information on running the following walkthrough, see **Running the Tutorial Programs (Section 7.2.1)**.

> **Note:** The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/SmartRefresh directory.

- **Creating the Backing Bean - Part 1 of 4 (Section 7.2.5.1.1)**
- **Creating the JSP Page - Part 2 of 4 (Section 7.2.5.1.2)**
- **Configuring the Application - Part 3 of 4 (Section 7.2.5.1.3)**
- **Deploying the Application - Part 4 of 4 (Section 7.2.5.1.4)**



Country: AUS. City: Sydney.

Choose a country : Australia

Choose a city : Sydney

Employee List

First Name	Last Name	Email	Phone Number
Helen	Bennet	helen.bennet@example.com	555 5556
Paul	Dundee	paul.dundee@example.com	555 4449
Hiroshi	Tannamurri	hiroshi.tannamurri@example.com	555 3388
Daniel	Tonini	daniel.tonini@example.com	555 1289
Phil	Cramer	phil.cramer@example.com	555 2269
Fiona	Mango	fiona.mango@example.com	555 4554

7.2.5.1.1 Creating the Backing Bean - Part 1 of 4

First, you need to create the backing bean that will be used to model the data and actions needed by the Web page. The Web page will contain four main components:

- an output text field to display the country and city selected
- a drop-down list for the list of countries
- a drop-down list for the list of cities
- a WebGrid component to display employee data

When a country is selected from the country drop-down list, the output text is updated to show which country was selected. In addition, the list of cities in the city drop-down list is updated. When a city is selected from the city drop-down list, the output text is updated to show which city was selected, and a grid is displayed showing a list of employee data for the selected country and city.

To model these components, we will create a Java class called SelectItemListBean.

Note: The full source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/SmartRefresh directory.

A member variable will be used to model the output text field, drop-down lists values, and *WebGrid*™ component as follows:

In Java:

```
/** Handle to the value selected in the city drop down list */
private UIInput city = null;

/** Handle to the value selected in the country drop down list */
private UIInput country = null;

/** Handle to the grid used to display the employee data */
private GridView employeeGridView = null;

/** Handle to the text field used to display the output text */
private UIOutput textOutput = null;
```

The values selected in the drop-down list are represented by the standard JSF UIInput class and are called "city" and "country". The text that displays the selected country and city is represented by a standard JSF UIOutput class. The WebGrid component is modeled using NetAdvantage for JSF’s gridView component and is called employeeGridView. In order to provide the Web page with access to these objects, the standard setters and getters are provided.

There are also three other get methods.

- getCountryList() method -- Used by the country drop-down list to retrieve a list of countries.
- getCityList() method -- Used by the city drop-down list to retrieve a list of cities.
- getEmployeeList() -- Used by the grid to retrieve a list of employees for the selected country and city.

In Java:

```
public List getCountryList()
{
  return SelectItemDAO.getCountryList();
}

public List getCityList()
{
  String[] countries = getSelectedCountries();
  List ret = new ArrayList();

  if(countries!=null)
  {
    for(int i=0; i<countries.length; ++i)
    {
      ret.addAll(SelectItemDAO.getCityList(countries[i]));
    }
  }
  return ret;
}

public List getEmployeeList()
{
  List employeeList=new ArrayList();
  String[] cities = getSelectedCities();

  if(cities!=null)
  {
    EmployeeDAO employeeDAO = new EmployeeDAO();

    for(int i=0; i<cities.length; ++i)
    {
      if(cities[i].equals("CA"))
      {
        employeeList.addAll(employeeDAO.getCamberraList());
        continue;
      }
      if(cities[i].equals("SY"))
      {
        employeeList.addAll(employeeDAO.getSydneyList());
        continue;
      }
      if(cities[i].equals("BR"))
      {
        employeeList.addAll(employeeDAO.getBristolList());
        continue;
      }
      if(cities[i].equals("LO"))
      {
        employeeList.addAll(employeeDAO.getLondonList());
        continue;
      }
      if(cities[i].equals("CH"))
      {
        employeeList.addAll(employeeDAO.getChicagoList());
        continue;
      }
      if(cities[i].equals("SF"))
      {
        employeeList.addAll(employeeDAO.getSanFranciscoList());
        continue;
      }
    }
  }
  return employeeList;
}
```

In order to react to the updates to the country and city drop-down lists, two methods need to be created so that they can be connected to the ValueChange vent for each drop-down list.

In Java:

```
public void onChangeCity(ValueChangeEvent event)
{
  // rebind the dataGrid
  this.employeeGridView.dataBind();
}

public void onChangeCountry(ValueChangeEvent event)
{
  // rebind the dataGrid
  employeeGridView.dataBind();
}
```

Both of these methods simply cause the grid to refresh its data based on the updates to the country or city.

In addition to the SelectItemListBean class, there are also three other classes needed for this example:

- Employee -- This class is a simple JavaBean used to represent the data associated with a single employee (first name, last name, e-mail address, and phone number).
- EmployeeDAO -- This class is used to represent a list of employees from a particular city. This class contains a static set of arrays of the employees and a getter for each of the possible cities.
- SelectItemDAO -- This class is used to represent the list of countries and cities listed in the two drop-down lists. This class has only the following two getters:
  - getCountryList() -- Retrieves the list of countries.
  - getCityList() -- Returns the list of cities of the selected country.

Related Topic

Creating the JSP Page - Part 2 of 4 (Section 7.2.5.1.2)

7.2.5.1.2 Creating the JSP Page - Part 2 of 4

This topic continues from **Creating the Backing Bean - Part 1 of 4 (Section 7.2.5.1.1)**.

The next step is to create a JSP™ page (smartRefresh.jsp) that will display the components and bind the values and actions to the backing bean.

> **Note:** The full JSP code, source code and resources needed to build and deploy this tutorial can be found in the *Install_Dir*/documentation/tutorials/SmartRefresh directory.

For the country drop-down list, we will use the Infragistics drop-down list as follows:

In JSP:

```
<ig:dropDownList id="countryDropDown"
        dataSource="#{webinput_dropDown.countryList}"
        binding="#{webinput_dropDown.country}"
        valueChangeListener="#{webinput_dropDown.onChangeCountry}"
        smartRefreshIds="cityDropDown,employeeGridView,textOutput"
        />
```

Where:

- dataSource="#{webinput_dropDown.countryList}" -- Binds the list of countries from the getCountryList() method of the SelectItemListBean class.
- binding="#{webinput_dropDown.country}" -- Binds the value of the selected item in the drop-down list to the country member variable of the SelectItemListBean class.
- valueChangeListener="#{webinput_dropDown.onChangeCountry}" -- Causes the onChangeCountry method of the SelectItemListBean class to be called whenever the value of the drop-down list changes.
- smartRefreshIds="cityDropDown,employeeGridView,textOutput" -- Is a comma-delimited list of other JSF components that get updated when the drop-down list is changed or updated. This means that each time the country drop-down list is updated, the city drop-down list will refresh its city list; the grid will be refreshed with the latest data; and the text displaying the selected country and city will be updated.

The city drop-down list uses the Infragistics drop-down list with similar arguments.

In JSP:

```
<ig:dropDownList id="cityDropDown"
        dataSource="#{webinput_dropDown.cityList}"
        binding="#{webinput_dropDown.city}"
        valueChangeListener="#{webinput_dropDown.onChangeCity}"
        smartRefreshIds="employeeGridView,textOutput"
        />
```

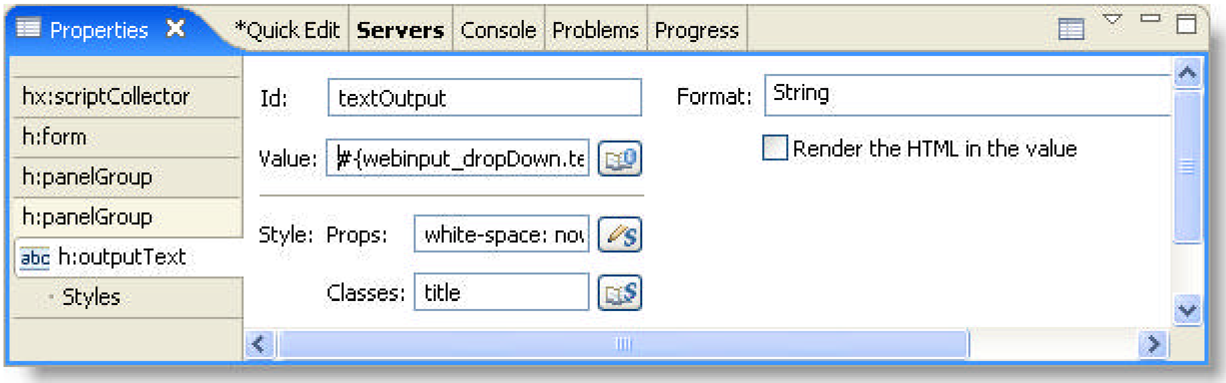
To display the employee data, Infragistics' WebGrid will be used. As shown in the JSP code below, WebGrid has two bindings. The first binding = "#{ webinput_dropDown.employeeGridView}" provides the backing bean with access to the WebGrid so that it can be updated when the country or city drop down lists are changed. The second binding dataSource= "#{webinput_dropDown.employees}" binds the employees list of the backing bean to WebGrid.

In JSP:

```
<ig:gridView id="employeeGridView"
        binding="#{webinput_dropDown.employeeGridView}"
        dataSource="#{webinput_dropDown.employeeList}">
        <f:facet name="header">
                <h:outputText value="Employee List" />
        </f:facet>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="First Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.firstName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Last Name" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.lastName}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Email" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.email}" />
        </ig:column>
        <ig:column>
                <f:facet name="header">
                        <h:outputText value="Phone Number" />
                </f:facet>
                <h:outputText value="#{DATA_ROW.phoneNumber}" />
        </ig:column>
</ig:gridView>
```

To create a JSP page using Smart Refresh at design time using IBM® Rational® Application Developer for WebSphere Software® 7.0:

1. From the palette, under the Enhanced Faces Component, drag the Output component to your page.
2. In the Properties window, select h:outputText .
3. Set the following properties:
   - Id -- textOutput
   - Value -- #{webinput_dropDown.textValue}



Id: textOutput

Value: #{webinput_dropDown.te

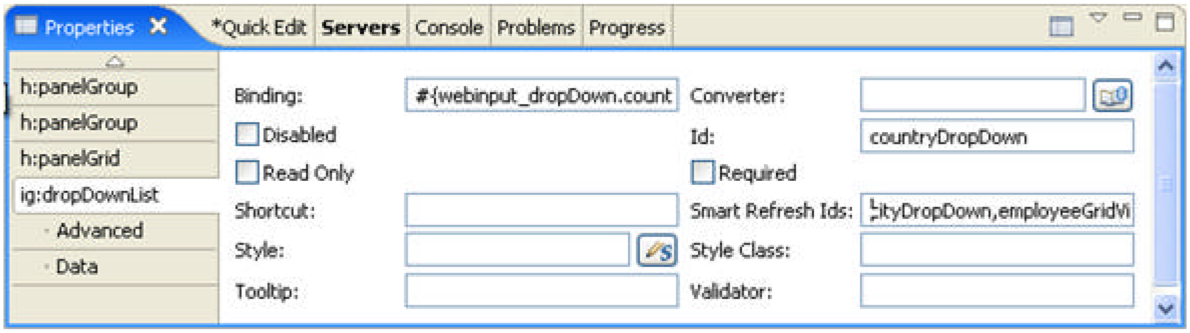
Format: String

□ Render the HTML in the value

Style: Props: white-space: no

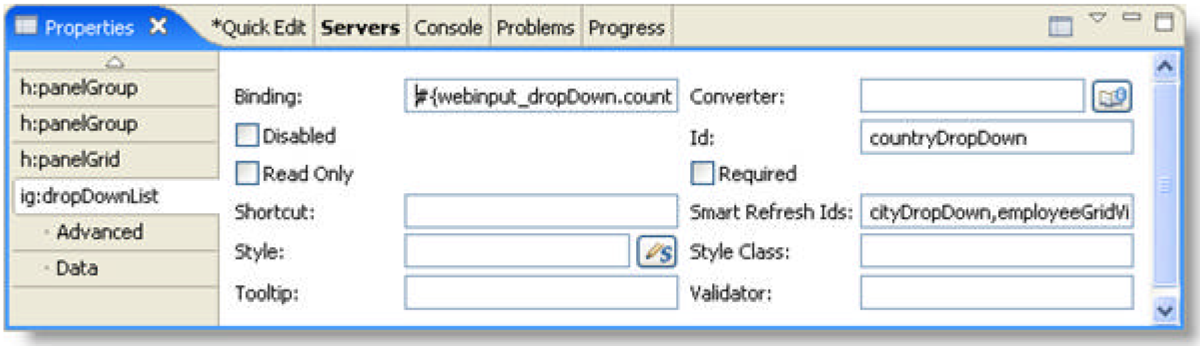
Classes: title

4. From the palette, under the Enhanced Faces Component, drag the Output component to your page.
5. In the Properties window, select h:outputText .
6. Set the Value to "Choose a country:".
7. From the palette, drag the DropDownList component to your page.
8. In the Properties window, select ig:dropDownList.
9. Set the following properties:
   - Binding -- #{webinput_dropDown.country}
   - Id -- countryDropDown
   - Smart Refresh Ids -- cityDropDown, employeeGridView, textOutput



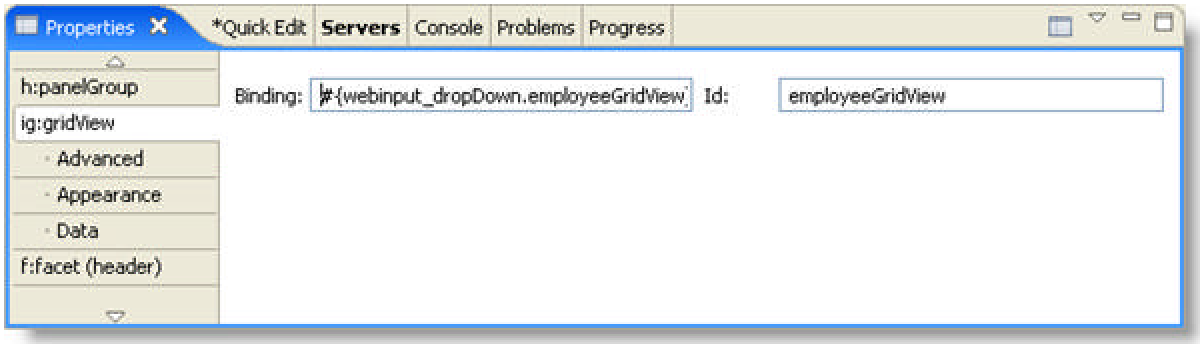
Binding: #{webinput_dropDown.count
□ Disabled
□ Read Only
Shortcut:
Style:
Tooltip:
Converter:
Id: countryDropDown
□ Required
Smart Refresh Ids: cityDropDown,employeeGridVi
Style Class:
Validator:

10. In the Properties window, under ig:dropDownList, select Data .
11. Set the following properties:
    - Data Source -- #{webinput_dropDown.countryList}
    - Value Change Listener -- #{webinput_dropDown.onChangeCountry}
12. From the palette, under the Enhanced Faces Component, drag the Output component to your page.
13. In the Properties window, select h:outputText .
14. Set the Value to "Choose a city:".
15. From the palette,drag the DropDownList component to your page.
16. In the Properties window, select ig:dropDownList.
17. Set the following properties:
    - Binding -- #{webinput_dropDown.city}
    - Id -- cityDropDown
    - Smart Refresh Ids -- employeeGridView, textOutput



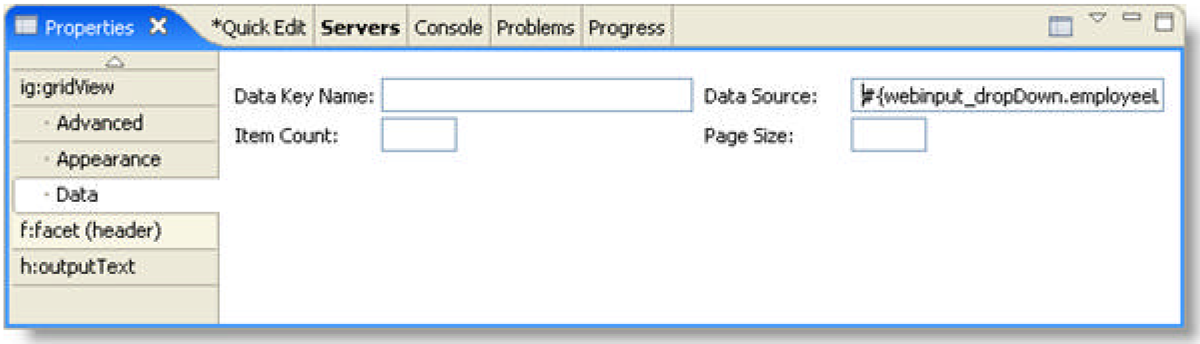
Binding: #{webinput_dropDown.count
□ Disabled
□ Read Only
Shortcut:
Style:
Tooltip:
Converter:
Id: countryDropDown
□ Required
Smart Refresh Ids: cityDropDown,employeeGridVi
Style Class:
Validator:

18. From the palette, drag the GridView component to your page.
19. In the Properties window, select ig:gridView.
20. Set the following properties:
    - Binding -- #{webinput_dropDown.employeeGridView}
    - Id -- employeeGridView



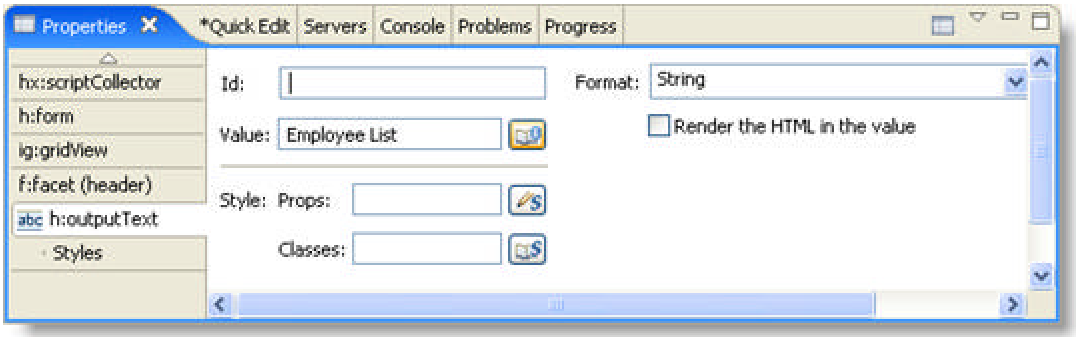
Binding: #{webinput_dropDown.employeeGridView} Id: employeeGridView

21. In the Properties window, under ig:gridView, select Data .
22. Set the following property:
    - Data Source: #{webgrid_dropDown.employeeList}



Data Key Name:
Data Source: #{webinput_dropDown.employeeL
Item Count:
Page Size:

23. In the Design view, select the Grid Header and set the following property:
    - Value -- Employee List

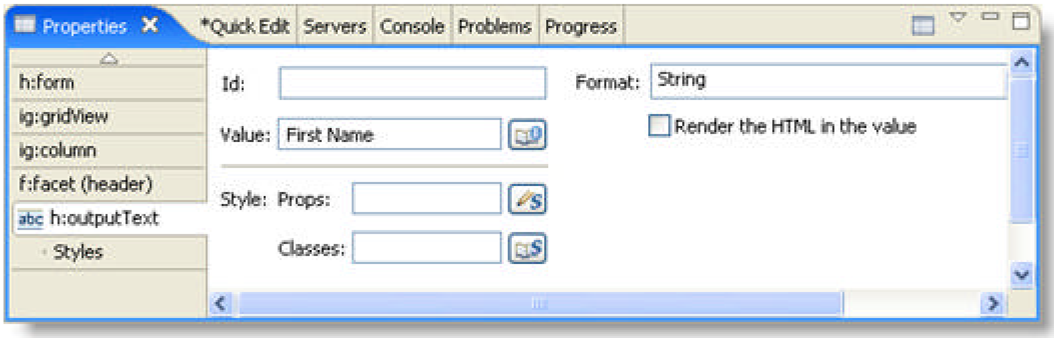


Id:
Value: Employee List
Format: String
□ Render the HTML in the value
Style: Props:
Classes:

24. The default GridView is created with two columns. From the palette, drag the Column component to your grid to add more columns.

25. In the Design view, for each column header, set the Value property:

- For Column Header 1, set Value to First Name.
- For Column Header 2, set Value to Last Name.
- For Column Header 3, set Value to Email.
- For Column Header 4, set Value to Phone Number.



Id:

Value: First Name

Format: String

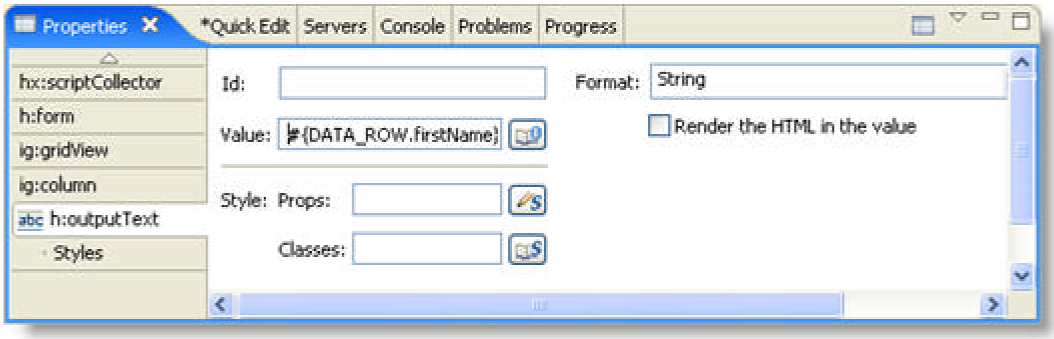
□ Render the HTML in the value

Style: Props:

Classes:

26. In the Design view, for each column, set the Value property:

- For Column 1, set Value to #{DATA_ROW.firstName}.
- For Column 2, set Value to #{DATA_ROW.lastName}.
- For Column 3, set Value to #{DATA_ROW.email}.
- For Column 4, set Value to #{DATA_ROW.phoneNumber}.



Id:

Value: #{DATA_ROW.firstName}

Format: String

□ Render the HTML in the value

Style: Props:

Classes:

27. Save your project.

Related Topic

Configuring the Application - Part 3 of 4 (Section 7.2.5.1.3)

7.2.5.1.3 Configuring the Application - Part 3 of 4

This topic continues from **Creating the JSP Page - Part 2 of 4 (Section 7.2.5.1.2)**.

Configuration of this application is relatively simple. Because this is a one-page application, the only files that need to be updated are the web.xml and the managed-beans.xml files. The other JSF configuration files (navigation.xml and faces-config.xml) do not need to be changed. For the web.xml file, the following entries need to be added to enable Faces Servlet and define several JSF parameters.

In XML:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup> 1 </load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <URL-pattern>*.faces</URL-pattern>
</servlet-mapping>

<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
</context-param>

<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/navigation.xml,/WEB-INF/managed-beans.xml</param-value>
</context-param>
```

The managed-beans.xml file must be updated to define the connection between the alias used in the JSP file and the SelectItemListBean backing bean. To do this, we add the following entry.

In XML:

```
<managed-bean>
  <description>
      Backing bean for the frames sample
  </description>
  <managed-bean-name>webinput_dropDown</managed-bean-name>
  <managed-bean-class>
      com.infragistics.faces.samples.webinput.shared.SelectItemListBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Related Topic

Deploying the Application - Part 4 of 4 (Section 7.2.5.1.4)

7.2.5.1.4 Deploying the Application - Part 4 of 4

This topic continues from **Updating the Configuration Files - Part 3 of 4 (Section 7.2.4.1.3)**.

Deploying the application is simply a matter of putting all the files in the correct location. Below is a brief description of the names and locations of the key files. The Apache® Ant build file that is included in the *Install_Dir*/documentation/tutorials/SmartRefresh directory contains tasks to create the WAR file and deploy it to an Apache Tomcat server. If your environment is different, you must update the build file.

- /resources -- Contains some of the default images used by the controls
- /resources/infragistics -- Contains the JavaScripts needed by the components.
- /resources/infragistics/themes -- Contains a set of theme subfolders. Each subfolder contains all the CSS files and image files required by the components. (The default theme is blue.)
- WEB-INF
    - /lib -- Contains all the JAR files that this application depends on, including a JAR file that contains the EmployeeDAO class
    - web.xml
    - managed-beans.xml
    - navigation.xml
    - faces-config.xml
- smartRefresh.jsp – The main page that contains the drop-down lists and employee data grid.
- index.jsp – The home page which simply redirects the user to the smartRefresh page through the Faces Servlet.