

Developer's Guide



SAP NetWeaver 2004s Running an Enterprise Portal

Document Version 1.70 – März 2006

**SAP AG**

Dietmar-Hopp-Allee 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

© Copyright 2005 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components.






Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Typographic Conventions

Type Style	Represents
<i>Example Text</i>	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
Example text	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Contents

1	GETTING INVOLVED.....	2
1.1	Portal Runtime	2
1.1.1	Portal Applications.....	2
1.1.2	How Requests Are Handled	16
1.1.3	Working with the PRT.....	26
1.1.4	Deployment of Applications	33
1.1.5	Deployment Policy (Enterprise Portal 5.0).....	33
1.1.6	Testing Components	33
1.2	Web Dynpro Applications for the Portal	33
1.2.1	Web Dynpro Java	33
1.2.2	Web Dynpro ABAP.....	33
1.3	Example.....	33
1.4	SAP NetWeaver Developer Studio Plug-In	33
1.4.1	Configuring the Plug-In.....	33
1.4.2	Managing Enterprise Portals	33
1.4.3	Managing PAR and JAR Files in the Project	33
1.4.4	Enterprise Portal Unit Test Studio Perspective	33
1.4.5	Enterprise Portal Web Services Checker Description.....	33
1.4.6	Development Configuration	33
1.4.7	Registering an Additional Plug-in.....	33
2	GO AND CREATE.....	33
2.1	Creating Your First Portal Application	33
2.1.1	Portal Runtime Basics	33
2.1.2	Creating the JSPDynPage.....	33
2.1.3	JSPDynPage Event Handling	33
2.1.4	Data Exchange between JSPDynPage and JSP	33
3	CORE DEVELOPMENT TASKS	33
3.1	Creating and Managing Content	33
3.1.1	Managing iViews and Other PCD Objects	33
3.1.2	Working with XML	33
3.1.3	Creating Administration Interfaces.....	33
3.1.4	Client-Side Eventing.....	33
3.2	Uniform Resource Identifier (URI).....	33
3.3	Uniform Resource Locator (URL).....	33
3.4	Uniform Resource Name (URN)	33
3.4.1	Page Builder.....	33
3.4.2	HTML-Business for Java	33
3.4.3	User Management Engine.....	33
3.5	Changes in the LoginModule Implementation	33
3.6	ServiceUserFactory	33
3.6.2	User Agent Service	33
3.7	Modifying the Desktop and Navigation.....	33

3.7.1	Navigating in the Portal	33
3.7.2	Creating Custom Layouts	33
3.7.3	Object-Based Navigation	33
3.8	Connecting to Backend Systems	33
3.8.1	Application Integrator	33
3.8.2	Connector Framework	33
3.8.3	Dynamic System Resolution	33
3.9	Specialities in the Portal	33
3.9.1	Implementing an External-Facing Portal	33
4	ENSURING QUALITY	33
4.1	Developing Well Performing Portal Applications	33
4.1.1	Server Side Programming	33
4.1.2	Java Programming	33
4.1.3	Portal Application Programming Model	33
4.1.4	Enterprise Portal Services	33
4.1.5	Database Access	33
4.1.6	Enterprise Portal Performance Ruleset for JLin	33
4.1.7	Checklist for Reviews	33
4.2	File Access	33
4.2.1	Ensuring Supportability with Metrics and Audits	33
4.2.2	References/Bibliography	33
4.3	General Rules and Guidelines for Managing Exceptions	33
5	REFERENCE	33
5.1	Portal APIs	33
5.2	Samples	33

Running an Enterprise Portal

Purpose

This section describes how to build applications for the portal. The guide explains how to build portal applications and how to make use of services provided by the portal or the underlying J2EE engine.

The guide is divided into the following sections:

- [Getting Involved \[Page 2\]](#): Describes basic concepts about the portal that you need in order to build portal applications. It also describes the SAP NetWeaver Developer Studio features that enable you to more easily build applications.

Portal Applications can be built using one of the following technologies:

- **Portal Runtime:** The [Portal Runtime \[Page 2\]](#) (PRT) is the underlining engine on which the portal runs. The PRT provides the infrastructure for creating PRT-based portal components, which generate HTML for display, and portal services, which provide general functionality to components and other services.
- **Web Dynpro:** This is the preferred method for building user interfaces. You can build Web applications with Web Dynpro, and then display the applications in the portal.

Web Dynpro is described in detail in [Web Dynpro for Java \[External\]](#) and [Web Dynpro for ABAP \[External\]](#). The best method for displaying Web Dynpro applications in the portal is described in [Web Dynpro Applications for the Portal \[Page 33\]](#).

- [Go and Create \[Page 33\]](#): Describes how to build a simple portal application using the PRT.
- [Developing Applications \[Page 33\]](#): Describes what you can do within a portal application with the help of built-in portal services.
- [Ensuring Quality \[Page 33\]](#): Describes the techniques for making sure that your applications are written properly.
- [Reference \[Page 33\]](#): Provides the Javadocs for portal and portal runtime APIs, as well as other vital information about the portal.

Prerequisites

The guide assumes that you have the basic knowledge of the following areas:

- Java and servlet programming.
- Basic operating system commands for navigating in the file system, copying and editing files.

Some portal services require knowledge of the following:

- JSP (Java Server Pages)
- HTML

1 Getting Involved

This section describes basic concepts about the development and deployment of portal applications, and includes the following:

- [Portal Runtime \[Page 2\]](#)
- [Web Dynpro Applications for the Portal \[Page 33\]](#)
- [SAP NetWeaver Developer Studio Plug-In \[Page 33\]](#)
-

1.1 Portal Runtime

Purpose

The Portal Runtime (PRT) provides a Java-based framework for running portal applications, which produce content for display in the portal.

The PRT is packaged as a J2EE Web application called the *Enterprise Portal Base Component* (EPBC).

The EPBC file contains three main parts:

- **Libraries and Configuration Files:** PRT and its configuration
- **Core Applications:** Core PRT services. These applications are not stored in the PCD but are deployed on each server node. Core applications have the following characteristics:
 - Provide core functionalities, such as authentication and PCD.
 - Loaded before other applications when the portal is initialized.
 - Never loaded into the Application Repository.

- **Portal Applications:** Basic portal applications shipped with the portal and stored in the Application Repository.

Regular applications are started only when required, either because the application is marked to be loaded on startup (by setting in the deployment descriptor) or because an element of the application has to be executed.

PRT Services

The PRT provides the following J2EE services:

- **Portal Runtime Container:** Manages the following runtime processes:
 - Deployment of portal applications
 - Communication between portal applications and J2EE applications.
- **PRTBridge:** Provides a link between portal runtime processes and the J2EE engine cluster configuration. It is installed as a J2EE engine service.

1.1.1 Portal Applications

A Portal Runtime application defines a collection of portal objects, which are of two types:

Portal Runtime

- **Portal Component:** Produces HTML output to be displayed in an iView in the portal, and is triggered by an HTTP request, as described in [Components \[Page 3\]](#).
- **Portal Service:** Provides a service, that is, some sort of processing but is not bound to a specific HTTP request, as described in [Services \[Page 3\]](#).

An application is a set of files – java classes, HTML, JSP and others – that define a set of components and services. The files are bundled and zipped into a single file with the extension `.par`. Such a file is known as a PAR file. Portal administrators upload PAR files in order to make the enclosed components and services available.

For more information on how a PAR file is constructed, see [PAR Files \[Page 7\]](#).

1.1.1.1 Components

Definition

A portal component produces HTML output to be displayed in an iView of the portal. The component can provide all the HTML for an iView, or can provide a fragment of HTML to be included within the HTML output of another component.

Content administrators create iViews from portal components. The HTML produced by the portal component is displayed in the iView. For more information on iViews, see the SAP NetWeaver documentation on the Help Portal (help.sap.com) → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *Content Administration* → [iViews \[External\]](#).

You create a portal component by doing the following:

- Write a java class that extends `AbstractPortalComponent`, which implements `IPortalComponent`.



Note: You could write your own class that implements `IPortalComponent`, but it is strongly advised to simply extend `AbstractPortalComponent`.

- Add the java class and any related files (such as HTML or JSP files) to a PAR file.
- Modify the deployment descriptor (`portalapp.xml` file) of the PAR file to provide information about the component, such as the name of the component, the name of the implementing class and configuration properties.

For more information on how a PAR file is constructed, see [PAR Files \[Page 7\]](#).

1.1.1.2 Services

Definition

A portal service provides processing that is not bound to a specific request. A service can be called by a component or by other services.

For example, the portal provides a transformation service for transforming XML, as well as the EPCF service for creating Javascript in an iView that enables client-side eventing between iViews.

A service cannot be personalized.

You create a portal service by doing the following:

- Write a java class that implements `IService`. The class includes the `IService` methods, such as `init()` and `destroy()`, as well as custom methods that you want to expose through the service.

You generally create a java class for a service with the help of the portal plug-in to NetWeaver Developer Studio, which creates generic implementations of the standard `IService` methods. For more information on the plug-in, see [SAP NetWeaver Developer Studio Plug-In \[Page 33\]](#).

- Add the java class to a PAR file.
- Modify the deployment descriptor (`portalapp.xml` file) of the PAR file to provide information about the service, such as the name of the service, the name of the implementing class and configuration properties.

For more information on PAR files, see [PAR Files \[Page 7\]](#).

1.1.1.3 Object Names, Aliases and References

This section describes how to refer to portal objects from within your component or service, and includes the following:

- [Object Names \[Page 4\]](#): Describes how to refer to applications, components and services.
- [Object Aliases \[Page 5\]](#): Describes how to create an alias for an application and service and how to use these aliases to refer to applications, components and services.
- [Object References \[Page 5\]](#): Describes how to retrieve a reference to an application resource, component or service from within a component or service, and how to add a reference to an another application so that your application can use the components and services of another applications.

1.1.1.3.1 Object Names

The following rules apply for the naming of portal objects:

- Applications are named for the PAR file that defines the application, without the `.par` extension.

For example, the application defined in `myApplication.par` is called `myApplication`.

- Services and components are named after the name of the application, followed by a period (`.`), followed by the name of the component or service as defined in the application's deployment descriptor.

For example, the name of the component called `myComponent` that is defined in `myApplication.par` is called `myApplication.myComponent`.

For more information on how to name components and services, see [Deployment Descriptor \(portalapp.xml\) \[Page 8\]](#).

1.1.1.3.2 Object Aliases

You can define an alias for an application and a service, generally for reasons of compatibility.

Aliases are defined in the `alias` attribute of the `<application>` or `<service>` element in the deployment descriptor.

The following rules apply to the use of aliases:

- An application alias can be used instead of the name of the application in all cases.
- A component cannot have an alias, but you can use the application alias in the component's fully qualified name.
- A service alias can be used instead of the name of the service. However, you must either use the service alias (without the application name or alias) or the service name preceded by the application name or alias.

Example

`myApplication.par` defines the following entries:

```
<application alias="myAlias">
  <services>
    <service name="myService" alias="myServiceAlias" />
  </services>
  <components>
    <component name="myComponent" />
  </components>
</application>
```

The following shows the valid names for each type of portal object defined above:

- **Application:** `myApplication` or `myAlias`
- **Component:** `myApplication.myComponent`, `myAlias.myComponent`
- **Service:** `myApplication.myService`, `myAlias.myService`, `myServiceAlias`

1.1.1.3.3 Object References

The following table shows how to access components, services and resources in the current application or in another application.

Object	How to Reference
Component	<p>A component can reference another component by retrieving an <code>IPortalComponentContext</code> object for the component from the <code>IPortalComponentRequest</code> object, as in the following example:</p> <pre>request.getComponentContext("myApplication.myComp");</pre> <p>The above example gets a reference to the <code>myComp</code> component defined in the <code>myApp</code> application. You can use the object, for example, to add a component to the POM tree, or to get information and resources from the component.</p>
Service	A component or service can reference a service by retrieving an <code>IService</code>

Portal Runtime

	<p>object for the service. The <code>IService</code> object can then be cast into the interface for the corresponding service.</p> <pre>PortalRuntime.getRuntimeResources().getService(ITransformerService.KEY);</pre> <p>The above example gets a reference to the transformation service. <code>PortalRuntime</code> is a member of the <code>com.sapportals.portal.prt.runtime</code> package.</p>
Resource	<p>A component can reference a Web resource packaged in its application or in another application by retrieving an <code>IResource</code> object for the resource from the <code>IPortalComponentRequest</code> object, as in the following examples:</p> <p>For a resource in the same application:</p> <pre>request.getResource(IResource.IMAGE, "images/myImage.gif");</pre> <p>For a resource in another application:</p> <pre>request.getResource("myApp", IResource.IMAGE, "images/myImage.gif");</pre> <p>A component can use a Web resource, for example, to create a link to the resource or to include it in its content.</p>
Property (component)	<p>A component can reference one of its profile properties, which are defined in <code><property></code> elements in the <code><component-profile></code> element in the component's deployment descriptor.</p> <p>You can also access properties defined by the semantic object in which the component is running. For example, if the component is run as an <code>iView</code>, the <code>iView</code> semantic object defines properties, such as <code>com.sap.portal.pcm.Title</code> for the <code>iView</code>'s name.</p> <p>The property value is first defined in the deployment descriptor, and then can be modified by an administrator for the specific <code>iView</code>, and then can be modified by the current user if the property is defined as personalizable.</p> <p>The following retrieves the value of <code>myProperty</code>:</p> <pre>IPortalComponentProfile profile = request.getComponentContext().getProfile();</pre> <pre>String propertyValue = profile.getProperty("myProperty");</pre> <p>The properties defined in the <code><component-config></code> element cannot be referenced.</p>
Property (service)	<p>A service can reference one of its profile properties, which are defined in <code><property></code> elements in the <code><service-profile></code> element in the component's deployment descriptor, within the service's <code>init()</code> method.</p> <p>The <code>init()</code> method is passed a <code>IServiceContext</code> object, which you can use as follows to retrieve a property value:</p> <pre>String propertyValue = serviceContext.getServiceProfile() .getProperty("myProperty");</pre>

Referencing Other Applications

To access components and services defined in another application, your application must reference the other application in a `<property>` element of the `<application-config>` element of its deployment descriptor.

For example, the following enables your application to make use of components and services in the application called `com.sap.myApp`:

```
<property name="SharingReference" value="com.sap.myApp"/>
```

The above reference is required in order to access components or services in `com.sap.myApp`, but not required for accessing the application's resources. For more information about `<property>` elements in the `<application-config>` element of the deployment descriptor, see [Application Configuration \[Page 9\]](#).

1.1.1.4 PAR Files

Portal applications are packaged in PAR (Portal Archive) files. A PAR file is a standard ZIP file with a `.par` extension that contains all Java classes, Web resources and other files required to run the application.

Within the portal, the name of the PAR file is the name of the application. The fully qualified name of any component or service defined in the PAR is the name of the PAR file, followed by a period (`.`) and then the name of the component or service.

The files contained in a PAR file are divided into two types:

- **Web resources**

All files in a PAR file that are accessible directly via an HTTP request, such as:

- HTML
- Images
- Stylesheets (`.css`)

The Web resources of a PAR also include a manifest file (`manifest.mf` in the `META-INF` folder), which provides version information. The folder and file are not required, but are recommended for tracking versions of your application.

Web resource files can be located anywhere outside the top-level `PORTAL-INF` folder.

You can create links to these files directly. For more information, see [Creating URLs to Web Resources \[Page 24\]](#).

- **Java Classes and Deployment Descriptor**

Java classes that implement the component and services defined in the PAR file, plus the deployment descriptor (`portalapp.xml`). The deployment descriptor specifies the components and services contained in the PAR file, and defines the configuration for the application, components and services. For more information the deployment descriptor, see [Deployment Descriptor \(portalapp.xml\) \[Page 8\]](#).

The PAR file's Java classes and deployment descriptor are located in the top-level `PORTAL-INF` folder.

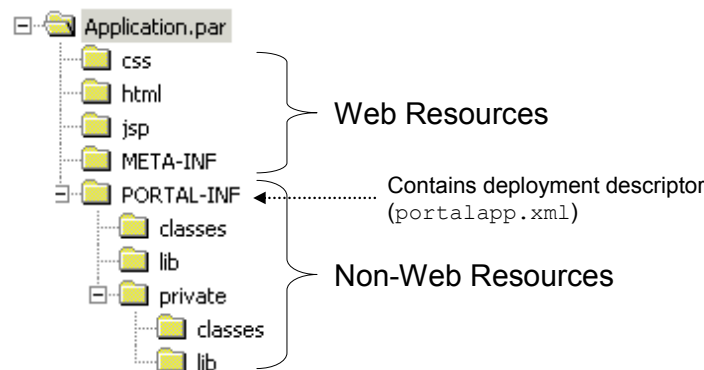
In the `PORTAL-INF` folder, the following folders have special purposes:

Folder	Description
--------	-------------

Portal Runtime

lib	Contains all application library files (JAR) that can be referenced by other portal applications.
classes	Contains all class files and other resources (such as resource bundles) that can be referenced by other portal applications.
private/lib	Contains all application library files (JAR) that are used exclusively by the application.
private/classes	Contains all class files and other resources (such as resource bundles) that are used exclusively by the application.

The following diagram provides an overview of the typical PAR file:



1.1.1.4.1 Deployment Descriptor (portalapp.xml)

The deployment descriptor of a PAR file is an XML file that defines the components and services contained in the PAR, including the initial configuration and the parameters that can be modified by administrators and users. The descriptor also contains configuration information for the entire application.



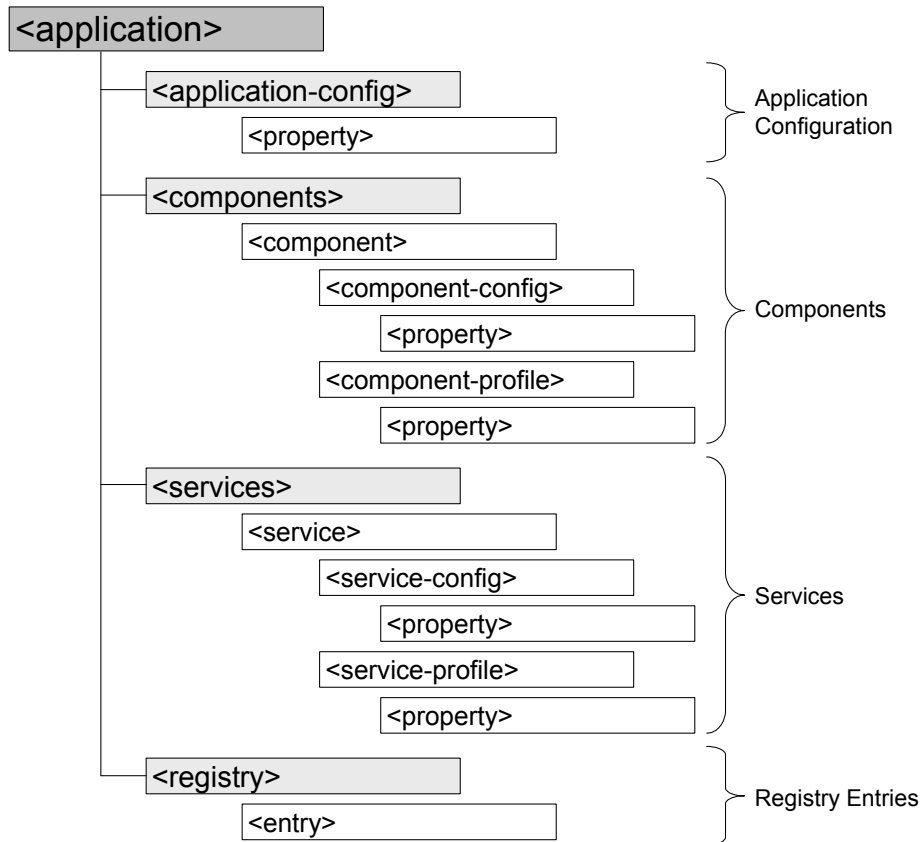
Even if you provide a Java class file for a component (or service), the component does not exist unless it is defined in the deployment descriptor.

The `<application>` element is the root element and is mandatory.

The following are the elements that you can define inside the `<application>` element:

- `<Application-Config>`: Defines configuration for the entire application, such as the application's security zone and whether the application should be loaded as soon as it is deployed.
- `<Components>`: Defines the components that are part of the application. This element specifies each component's implementation class and safety level, as well as configurable parameters.
- `<Services>`: Defines the services that are part of the application. This element specifies each service's implementation class and safety level, as well as configurable parameters.
- `<Registry>`: Defines entries in the portal registry.

The following illustrates the structure of the deployment descriptor:



Configuration for the application, each component and each service is specified by `<property>` elements located within the appropriate element. For example, a component can be configured by adding `<property>` elements to the `<component>` elements for the component you want to configure.

Each `<property>` elements contains `name` and `value` attributes.

1.1.1.4.1.1 Application Configuration

The `<application-config>` element specifies the configuration for the entire application. It may contain an unlimited number of property elements.

The following is an example of the `<application-config>` element.

```

<application-config>
  <property name="SharingReference"
    value="com.sap.portal.contentproviders"/>
  <property name="releasable" value="false"/>
  <property name="startup" value="true"/>
  <property name="Vendor" value="sap.com"/>
  <property name="SecurityArea" value="NetWeaver.Portal"/>
</application-config>
  
```

The following are standard properties for an application.

Property	Value	Description
----------	-------	-------------

Portal Runtime

<code>PrivateSharingReference</code>	List of portal application names or aliases, separated by commas (,)	Enables references to other portal application APIs that you want to call from this application's (non-public) implementation.
<code>releasable</code>	<code>true</code> (default), <code>false</code>	Indicates whether the system can release instances of this application when the system runs low on memory. Since applications can be deployed during runtime, all applications can be released and application instances can be dropped at any time by the system.
<code>SecurityArea</code>	Any valid string	String identifying the security area for the application, for example, <code>NetWeaver.portal</code> . This property, along with the <code>Vendor</code> property, are used to define the security zone for this application. For more information on security zones, see Permission Model [Page 26] .
<code>ServicesReference</code>		Equivalent to <code>SharingReference</code> .
<code>SharingReference</code>	List of portal application names or aliases, separated by commas (,)	Enables references to other portal application APIs that you want to call from this application's API definition.
<code>startup</code>	<code>true</code> , <code>false</code> (default)	If set to <code>true</code> , the application is initialized on startup of the J2EE engine. This results in the application being deployed locally at startup.
<code>Testable</code>	<code>true</code> , <code>false</code> (default)	If set to <code>true</code> , indicates to the PAR Unit Test Studio that the application contains testable entities. For more information, see Testing Components [Page 33] .
<code>Vendor</code>	Any valid string	String identifying the company or organization that provided the application, for example, <code>sap.com</code> .

1.1.1.4.1.2 Components

The `<components>` element defines the portal components that are part of this application, and includes configuration information for each. The element includes one or more `<component>` elements, each of which defines one portal component.

The following is an example of a `<components>` element:

```
<components>
  <component name="CP_SEARCH">
    <component-config>
      <property name="ClassName"
value="com.sap.portal.MyComponent"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name="url" value=""/>
      <property name="transformers.pipe" value=""/>
    </component-profile>
  </component>
</components>
```

Each `<component>` element contains a `name` attribute, which defines the name of the component.

Each `<component>` element contains the following elements:

- `<component-config>`: Provides static configuration that cannot be changed by an administrator or personalized by an end user.

The following are standard properties for component configuration:

Name	Description
AuthRequirement	<p>The initial minimal authentication needed to execute the component. This property is supported for backward compatibility only.</p> <p>The property can be one of the following values:</p> <ul style="list-style-type: none"> • User: Authenticated user • Admin: Administrator • None: Anonymous user • <role list>: Users with the listed role
ClassName	The name of the implementation class of the portal component.
ComponentType	<p>The type of component, which can be one of the following:</p> <ul style="list-style-type: none"> • <code>servlet</code> • <code>jspnative</code> <p>The property is not required. If blank, an implementation of <code>IPortalComponent</code> must be specified by the <code>ClassName</code> property.</p>
JSP	If <code>ComponentType</code> is set to <code>jspnative</code> , this property specifies the path of the JSP page that implements the portal component. This path is relative to the <code>PORTAL-INF</code> folder).

Portal Runtime

ResourceBundleName	<p>The name of the component's resource bundle, which is used to store localized text strings.</p> <p>The resource bundle should be placed in the <code>/PORTAL-INF/private/classes</code> folder.</p> <p>Specify the resource without the <code>.properties</code> extension but include the file in the PAR with the extension.</p>
SafetyLevel	<p>The safety level for the component.</p> <p>For more information, see Permission Model [Page 26].</p>

- `<component-profile>`: Provides configuration that can be changed by an administrator or personalized by an end user. For each property, different values can be set for each iView derived from the component, and for each end user (if personalization is permitted).

The following are standard properties for a component profile:


Name	Description
ALLOW_BROWSER	<p>Indicates whether the portal uses the browser cache to render the component. The value can be one of the following:</p> <ul style="list-style-type: none"> • Yes: The content is stored in the browser cache and in the PRT cache. • No: The content is stored in the PRT cache, but not in the browser cache. • BrowserCache: The content is stored in the browser cache, but not in the PRT cache.
CachingLevel	<p>The scope of the cache, which can be one of the following:</p> <ul style="list-style-type: none"> • Shared: The cache content is shared among all users. • User: The content is private for each user. • Session: The content of the component is cached for the (servlet) session, whether or not a user is connected. This level is used to cache information related to the browser's sessions.
EPCFLevel	<p>Indicates whether the enterprise portal client framework (EPCF) is used.</p> <p>If two or more embedded iViews with different <code>EPCFLevel</code> values are rendered on a page (within the same frame), the effective level is computed as the maximum of all <code>EPCFLevel</code> values on the page.</p> <p>The property can have the following values:</p> <ul style="list-style-type: none"> • 0: This component does not use EPCF. Neither scripts nor applets are included. • 1: This component uses all EPCF features implemented with pure JavaScript. • 2: This component uses all EPCF features implemented with pure JavaScript and applets. <p>The default value is level 2.</p>

Portal Runtime

ValidityPeriod	The length of time, in milliseconds, that the cache is valid.
----------------	---

The component can also define custom parameters.

Each `<component-profile>` property can have the following sub-properties that define the property's personalization behavior:

Name	Description
<code>description</code>	A text token that can be resolved by the component's resource bundle. The locale-sensitive descriptive text for the property can be used in user interfaces.
<code>inheritance</code>	<p>Indicates whether any changes in this property within a descendent object is recognized by the Portal Runtime.</p> <ul style="list-style-type: none"> • final: Changes to the property value are not recognized by the Portal Runtime • non-final: Changes to the property value are recognized by the Portal Runtime (default). <p>Not matter the setting of the <code>inheritance</code> sub-property, a descendent object can store a different value for the property. The <code>inheritance</code> sub-property determines whether this value has any effect.</p> <p>A descendent object may have a different value for the property, but with the <code>inheritance</code> sub-property set to <code>final</code>, this value has no effect. If the <code>inheritance</code> sub-property is changed to <code>non-final</code>, the changed value in the descendent object would then have an effect.</p>  <p>You cannot set the <code>inheritance</code> sub-property to <code>final</code> in an <code>iView</code> unless that <code>iView</code> specifies a value for the property.</p>
<code>personalization</code>	<p>Indicates whether this property can be personalized. The value can be one of the following:</p> <ul style="list-style-type: none"> • none: No personalization • no-dialog: Personalization is enabled, but only through code. The property does not appear in the standard personalization window. • dialog: Personalization is enabled and the property appears in the standard personalization window. <p>The default is <i>dialog</i>.</p>
<code>plainDescription</code>	A description of the property for use in user interfaces, which is used if no <code>description</code> property is specified.

Portal Runtime

type	<p>Specifies the type of property. This value determines what types of controls are displayed for modifying this property.</p> <p>The value can be one of the following:</p> <ul style="list-style-type: none"> • select[<option>(,<option>)] • boolean <p>If blank, the property is treated as a string and edited in a textbox.</p>
------	---

These sub-properties have no meaning to the Portal Runtime. They are used during personalization or at design time by administration tools.

The following is an example of the declaration of a property named `Color` with sub-properties `personalization` and `type`:

```
<property name="Color" value="red">
  <property name="personalization" value="dialog"/>
  <property name="type" value="select[red,green]"/>
</property>
```

1.1.1.4.1.3 Services

The `<services>` element defines the portal services that are part of this application, and includes configuration information for each. The element includes one or more `<service>` elements, each of which defines one portal service.

The following is an example of a `<services>` element that defines one service.

```
<services>
  <service name="ContentProvider" alias="MyServiceAlias">
    <service-config>
      <property name="className"
value="com.sap.portal.MyService"/>
      <property name="startup" value="false"/>
    </service-config>
    <service-profile>
      <property name="MyProperty " value="true"/>
    </service-profile>
  </service>
</services>
```

Each `<service>` element can contain the following attributes:

- **Name:** The name of the service
- **Alias:** An alias for the service that can be used instead of the fully qualified name. This attribute is optional.

Each `<service>` element contains a `<service-config>` element that provides configuration for the service.

The following are standard properties for service configuration.

Name	Description
------	-------------

Portal Runtime

ClassName	The name of the implementation class of the portal service. This class must implement the <code>IService</code> interface.
startup	If set to <code>true</code> , the service is started at when the portal is started. If set to <code>false</code> or left blank, the service is started on demand.

Each `<service>` element can also contain a `<service-profile>` element that provides custom properties that can be modified by the system administrator.

There is only one set of values for these properties of a service.

1.1.1.4.1.4 Registry

The `<registry>` element defines any entries to be added or modified in the registry.

The following is an example of a `<registry>` element:

```
<registry>
  <entry path="/runtime/transformers" type="subcontext"/>
</registry>
```

The `<registry>` element can contain one or more `<entry>` elements, each of which defines one of the following:

- **Registry Entry:** A key-value pair. The entry is placed in the path specified by the registry subcontext specified by the `path` attribute. The key is the final part of the `path` attribute and the value is specified by the `name` attribute.

For example, the following creates the key `myTransformer` with the value `TransformerProvider` in the subcontext `runtime/transformers`:

```
<entry path="runtime/transformers/myTransformer"
  name="TransformersProvider" type="service"/>
```

Any other values specified in the entry are attributes of the registry entry. In the example above, `type` is an attribute of the `TransformersProvider` entry with the value `service`.

- **Registry Subcontext:** A new subcontext to be created in the existing registry subcontext specified by the `path` attribute. The `type` attribute must be `subcontext`.

The following creates the subcontext `transformers` in the `runtime` registry subcontext:

```
<entry path="/runtime/transformers" type="subcontext"/>
```

1.1.1.4.1.5 Deployment Descriptor Example

The following is an example of a deployment descriptor (`portalapp.xml`) file for an application that contains one portal service and one portal component.

```
<application>

  <application-config>
    <property name="releasable" value="true"/>
  </application-config>

  <components>
    <component name="ChatRoom">
      <component-config>
        <property name="ClassName"
```



```

        value="com.sap.portal.exampleapp.impl.ChatRoom"/>
    </component-config>
    <component-profile>
        <property name="diplayHistory" value="10">
            <property name="plainDescription" value="No. of messages"/>
            <property name="personalization" value="dialog"/>
        </property>
        <property name="diplayStyle" value="list">
            <property name="type" value="select[list,history]"/>
            <property name="personalization" value="dialog"/>
        </property>
    </component-profile>
</component>
</components>

<services>
    <service name="ChatService">
        <service-config>
            <property name="startup" value="true"/>
            <property name="className"
value="com.sap.portal.exampleapp.impl.ChatService"/>
        </service-config>
        <service-profile>
            <property name="chatHistory" value="100"/>
        </service-profile>
    </service>
</services>

</application>

```

1.1.2 How Requests Are Handled

The Portal Runtime is an HTTP servlet (called *pvt*) contained in a J2EE application (called *irj*), which runs in the SAP J2EE Web container. The servlet is the entry point for all requests, which are handled by the following components of the Portal Runtime:

- **Dispatcher:** This is the *pvt* servlet that receives the request. It selects the appropriate connection based on the request parameters, and passes the request to this connection.
- **Connection:** The connection does the following:
 - Treats servlet request and forwards it to the request manager (for the standard Portal Runtime connection, called *portal*).
 - Creates portal request and response objects for the request.
 - Manages user authentication
 - When requested, creates URLs for this connection.
 - Defines the hooks for the request. For more information on Hooks, see [Hooks \[Page 24\]](#).
- **Request Manager:** The request manager does the following:
 - Builds the POM tree and processes POM and request events.
 - Executes the portal components involved in the request and retrieves the content.

- **Object Broker:** The object broker manages the creation and lifecycle of all portal component and service objects. It manages all class loading and dependency issues.

Developers of portal applications have the greatest control in what occurs within the request manager, especially what occurs during the creation of the POM tree and content, as described in [Portal Object Model \(POM\) \[Page 17\]](#).

1.1.2.1 Portal Object Model (POM)

In order to generate the HTML content for a specific HTTP request, the portal creates a POM (Portal Object Model) tree of all components that will be involved in generating HTML for the request.

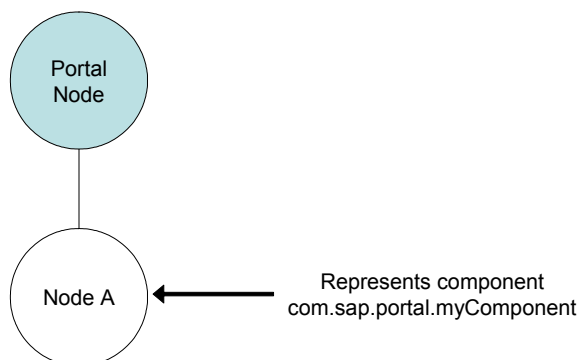
Often, there is only a single component for generating the content. But a component can delegate to another component the task of generating the HTML for the request, or can include the HTML from another component in its response. These are done during POM creation.

During the creation of the POM, the Portal Runtime calls methods of the components involved in the request. Afterward, during content creation, the Portal Runtime calls other methods of the components. These methods are described in Event Cycle.

POM Structure

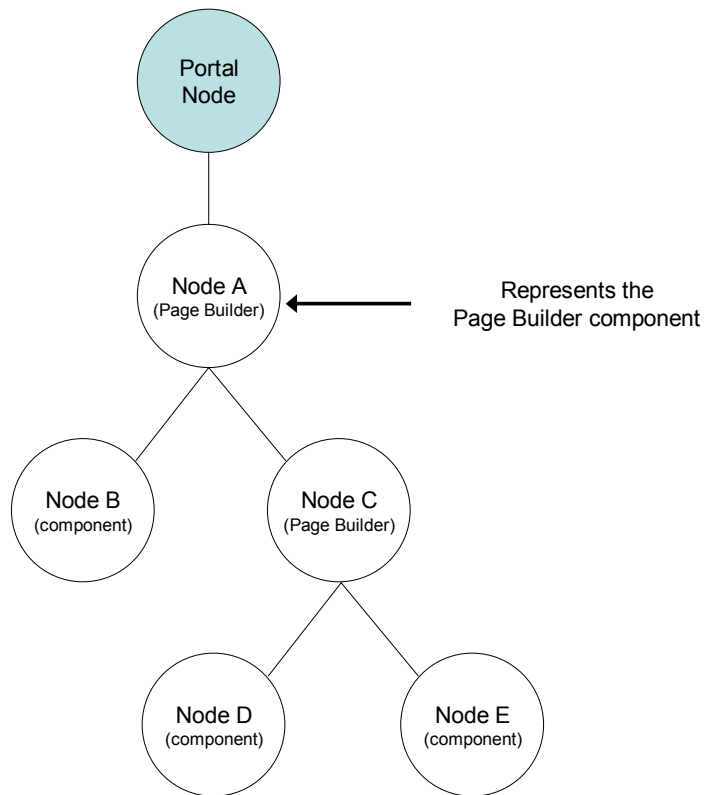
A node called the Portal Node is always the root of the POM, and is created automatically. The first child of this node is a node that represents the component specified in the request URL. This component can then include other components in the request by creating other nodes within the POM.

The following is a simple POM with one component, which could represent a request for a single iView that is rendered using the component `com.sap.portal.myComponent`:



Portal Runtime

The following is a more complex POM with several components involved in the request:

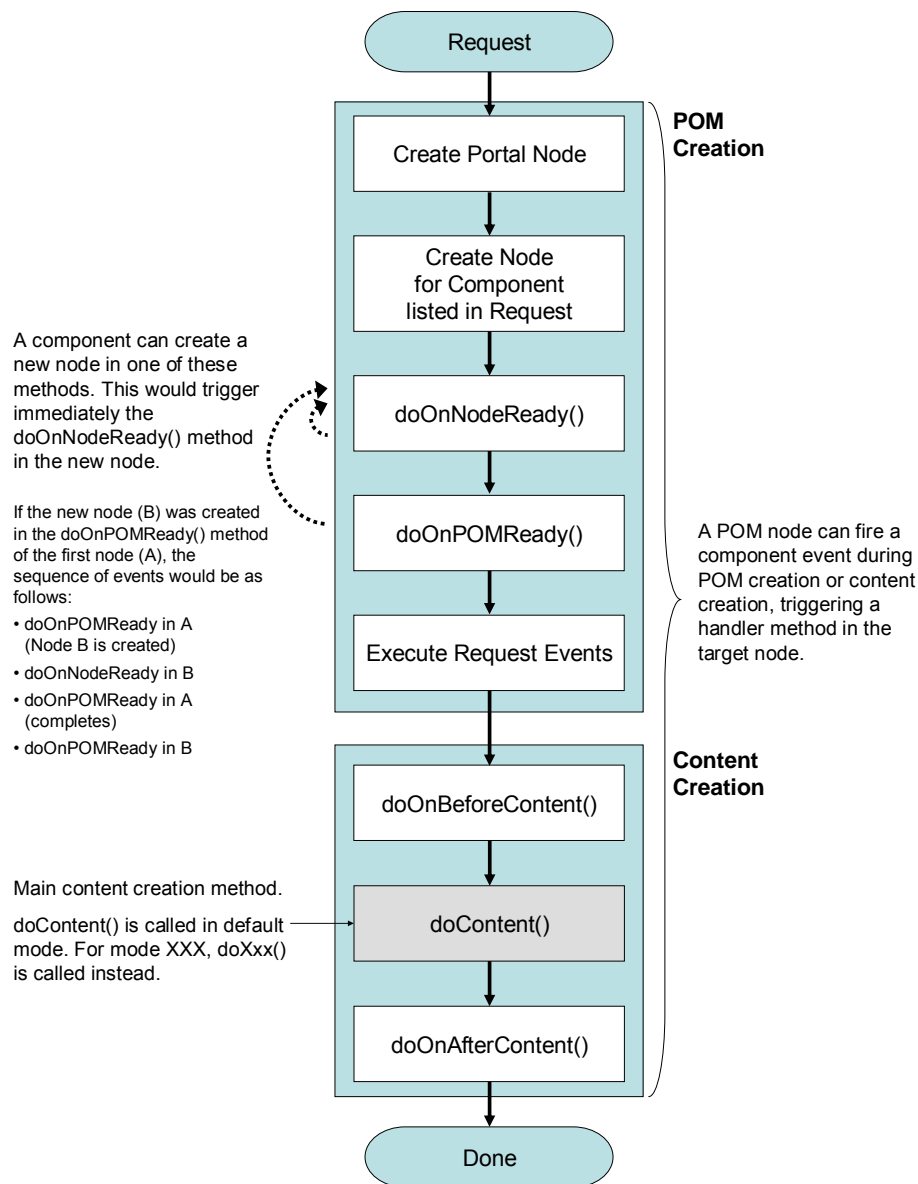


The above tree represents a page that includes an iView (component) and another page, which includes two iViews (components). The Page Builder component is a major user of the POM mechanism.

1.1.2.2 Request Flow

The following illustrates the POM process flow for handling a request:

Portal Runtime



The following is the sequence of method calls during POM creation and content creation:

- **`doOnNodeReady()`**: Called on a component when the node representing the component is added to the POM tree. Called once for each component.
- **`doOnPOMReady()`**: Called on each node when no more nodes are to be added to the POM tree. The Portal Runtime traverses the tree from top to bottom, calling the method for each component.

If a node is created in a `doOnPOMReady()` method, the new node's `doOnNodeReady()` method is called and then the `doOnPOMReady()` method finishes execution. The `doOnPOMReady()` method of the new node is then called.

- **Request Event Handlers**: Called if there are request events for special nodes.

Portal Runtime

- **doBeforeContent()**: Called just before content creation. The Portal Runtime traverses the tree from bottom to top, calling the method for each component.
- **doContent()**: Called to retrieve content from the top-most component in the POM tree. `doContent()` is called only in default mode. An alternative method is called for other modes. See Modes.

It is up to this mode to invoke the `doContent()` (or alternative method) of any other nodes in the POM tree.
- **doAfterContent()**: Called on each node when that component's `doContent()` (or alternative method) is finished.

Calls to `doOnNodeReady()`, `doOnPOMReady()` and request event handlers are part of the POM creation phase. Calls to `doBeforeContent()`, `doContent()` (or an alternative method if the mode is not the default mode) and `doAfterContent()` are part of the content creation phase.

1.1.2.2.1 Including Another Component's Content

During content creation, a component can include in its response the content from another component, as shown below:

```
INode childNode = request.getNode().getFirstChild();
response.include(request, childNode);
```

The component must be represented by a node in the POM tree.

1.1.2.2.1.1 Creating and Adding Nodes

To add a component node, do the following:

1. Create a portal component context for the component.

```
IPortalComponentContext portalContext =
    request.getComponentContext("myApp.myComp");
```

2. Create a component node.

```
IComponentNode componentNode = request.getNode().getPortalNode()
    .createComponentNode("myNode", portalContext);
```

3. Add the node to the POM tree. For example, the following adds a node to the portal node:

```
request.getNode().getPortalNode().addChildNode(componentNode);
```

Removing a Node

To remove a child node, a node can call `removeChildNode()` and supply a reference to the node to be removed.

1.1.2.3 Modes

A portal component's mode determines which method is called during content creation.

If no mode is specified, the component is rendered in the default mode, which causes the component's `doContent()` method to be called. For any other mode named `xxx`, the component's `doXxx` method is called.

For example, in help mode, the `doHelp()` method is called instead of `doContent()`.

For a component or for all components, you can designate a delegate component for handling any request for a specific mode. For more information, see [Delegation \[Page 21\]](#).

How the Mode is Set

The mode can be set in the following ways:

- **Request URL:** The request URL may contain a parameter named `prtmode` whose value is the mode.
- **Manually During POM Creation:** During the creation of the POM, you can set the mode of a specific node with the `setNodeMode()` method of the `INode` interface.

`AbstractPortalComponent` implements the methods for some of the built-in modes, such as `edit` and `help` mode (with `doEdit()` and `doHelp()` methods), but does not implement `doContent()` for the default mode. You must implement this mode.

1.1.2.3.1 Delegation

Instead of implementing a method within your portal component for a specific mode, you can specify another component to be called when your component is called in a specific mode.

This delegation is specified in the component's deployment descriptor:

```
<component name="mycomp">
  <component-config>
    <property name="ClassName" value="com.sap.MyComponent"/>
    <property name="mode" value="edit">
      <property name="delegate" value="DelegateComp.default"/>
    </property>
  </component-config>
</component>
```

When delegating for `xxx` mode, the `doXxx()` method is called on the delegate component. In the above example, `doEdit()` is called on `DelegateComponent.default`.

The subsequent request events are also sent to the delegate component until the mode is set to default.

Mode Delegation

You can create a component that serves as the delegate component for all components that are called in a specific mode.

To specify a delegate, add a registry entry with the name of the delegate component in a subcontext named for the mode, and place this subcontext under the `/runtime/prt.modes` subcontext.

The following example shows how to specify the `myApp.adminDelegate` component as the delegate for all components that are called in `admin` mode:

```
<registry>
  <entry path="/runtime/prt.modes/admin"
    name="myApp.adminDelegate"
    type="component" rebind="false"/>
```

```
</registry>
```

1.1.2.4 Events

You can define, fire and handle events during a portal request. The portal defines the following types of events:

- **Request Events:** Events defined in the request URL. For example, you may have a component that displays a form that gets submitted to the same component, but with a slightly modified URL that includes a request event that indicates that the *Submit* button was clicked.

For a request event named `xxx`, the portal tries to call

`doXxx(IPortalComponentRequest, IPortalRequestEvent)` prior to content creation.

If no such method exists, `doRequestEvent(IPortalComponentRequest, IPortalRequestEvent)` is called. The `IPortalRequestEvent` object contains the name of the event and any data.

For information on generating a URL with a request event, see [Creating URLs to Components \[Page 23\]](#).

- **Portal Object Model (POM) Events:** Predefined events that occur during POM creation and content creation. For a list of events, see [Request Flow \[Page 18\]](#).
- **Component Events:** A POM node can fire an event to other nodes in the POM. The following is an example of firing a component event:

```
IPortalNode myPortalNode = request.getNode().getPortalNode();
IPortalComponentEvent myEvent =
    myPortalNode.createPortalComponentEvent("myEvent");
myPortalNode.fireEventOnNode(myEvent, myNode);
```

For a component event named `XXX`, the portal tries to call

`doXxx(IPortalComponentRequest, IPortalRequestEvent)` immediately.

If no such method exists, `doComponentEvent(IPortalComponentRequest, IPortalComponentEvent)` is called. The `IPortalComponentEvent` object contains the name of the event and any data.

A node can filter the events it receives and choose to accept or reject each event. The method `accept()` is called on the node component before event handling occurs. If `accept()` returns `true`, the event is handled as usual; if `false` is returned, the event is ignored.

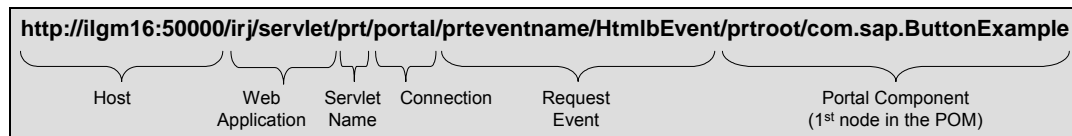
1.1.2.5 Request URL

This section describes how the request URL is constructed.



The format of portal URLs may change. Therefore, to create a URL, use the Portal Runtime API, as described in *Creating Request URLs*.

The following is a sample URL, which is relevant when using the standard `portal` connection.



A portal request URL generally contains the following parameters:

- **Web Application** (*irj*): The name of the J2EE Web application that contains the Portal Runtime.
- **Servlet Name** (*prt*): The name of the servlet that handles the request. This is the dispatcher servlet.
- **Connection** (*portal*): The connection for this request. The default connection for Portal Runtime applications is *portal*. There are special connection objects (for example, for WSRP or Knowledge Management requests).
- **Root Component** (*prtroot*): The root component that is being requested.
- **Mode** (*prtmode*): The mode of the request.
- **Request Event** (*prteventname*): The name of a request event to be fired for this request.
- **Request Event Target** (*prttarget*): The node in the POM that receives the request event.

The web application, servlet and connection parameters are single values (with the standard value shown in parentheses).

The root component, mode, request event and target parameters are key-value pairs (with the key shown in parentheses).

1.1.2.5.1 Creating URLs to Components

You may want to create links between portal components by putting the URL of a portal component in a hyperlink tag (<a>). The Portal Runtime includes an API for creating URLs to portal components.



Never hardcode URLs. Always use the API to create the URLs.

Procedure

1. Create a `IPortalComponentURI` object from the request object. This object is a helper class in creating the URL.

```
IPortalComponentURI componentURI = request.createPortalComponentURI();
```

2. Set the name of the component. In the example below, the link is to the portal component `myApplication.myComponent`.

```
componentURI.setContextName("myApplication" + "." + "myComponent");
```

3. Set the request event, if necessary.

```
IPortalRequestEvent myRequestEvent = request.createRequestEvent("myEvent");
componentURI.setPortalRequestEvent(myRequestEvent);
```

You can add parameters in the request by adding an `IPortalRequestEventData` object – which contains key-value pairs – to the `IPortalRequestEvent` object.

Portal Runtime

4. Set the target of the request event, if necessary. In the example below, the target is the current component.

```
componentURI.setTargetNode(request.getNode());
componentURI.setPortalRequestEvent(myRequestEvent);
```

5. Set the mode, if necessary.

```
componentURI.setNodeMode(NodeMode.EDIT_MODE);
```

6. Create the URL string with the following:

```
componentURI.toString();
```

1.1.2.5.2 Creating URLs to Web Resources

This section explains how to create links to Web resources contained in your portal application. Web resources are the files in your application that are accessible directly via an HTTP request.

The following code retrieves the URL to the XML file `BugsReminder.xml` and outputs it to the response:

```
IResource myResource =
    request.getResource(IResource.XML, "xml/BugsReminder.xml");
response.write(myResource.getResourceInformation().getURL(request));
```



Never hardcode URLs. Always use the API to create the URLs.

1.1.2.6 Hooks

The Portal Runtime has defined some points during the request cycle where you can add your own processing. Those entry points, which are called hooks, are defined by the connection and are run for every request.



Hooks affect every request to the portal, and are designed for special situations. Because hooks can have wide-ranging and dangerous side-effects, it is strongly recommended not to implement new hooks.

The following types of hooks can be defined (for the portal connection only):

- **POM Hooks:** Executed whenever a POM node is created.
- **Event Hooks:** Executed before any event is fired. It could replace the event, forward it to another object or cancel the publishing.
- **Document Hooks:** Executed before and after construction of the HTML document.
- **Response Hooks:** Executed before and after the `service()` method of the component. The hook can substitute the original response and is notified when the component has finished using the response object.
- **Component Hooks:** Executed before the `service()` method of the component. It could add content or replace the content.

The hooks to be executed are defined in the portal registry under `/runtime/hooks`.

1.1.2.7 Portal Registry

Purpose

The portal registry stores configuration information for the Portal Runtime. This information is for use by the Portal Runtime, and you should not modify the registry.

However, some applications may require you to create a registry entry. For example, if you create a new XML transformer, you must register the transformer by adding a registry entry at `/runtime/transformers`. A registry entry can be created by adding an `<entry>` element in your application's deployment descriptor.

For more information on creating registry entries, see [Registry \[Page 15\]](#).

Viewing the Registry

To view the registry, run the Portal Registry Browser by clicking *System Administration* → *Support* → *Portal Runtime* → *Portal Registry Browser*. The following is displayed:

Name	Class name
com.sapportals.portal.pcm.registeredServices	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext
broker	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext
UME	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext
com.sap.portal.appintegrator	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext
WP	com.sapportals.portal.prt.core.resource.MultiPropertiesResource
ContentCatalog	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext
Navigation	com.sapportals.portal.prt.jndisupport.util.MemoryHierarchicalContext

The **Name** column displays either a subcontext, in which it is displayed as a link, or the name of a registry entry. For a registry entry, the **Class name** column displays the value of entry.

For example, to view registered transformers, click the `runtime` subcontext, and then click the `transformers` subcontext. The registry entries in the **Name** column represent the names of transformers, and the value for each in the **Class name** column represents the service that implements the transformer.

Portal Runtime

JNDI environment settings:

Initial context factory: **com.sapportals.portal.prt.registry.PortalRegistryFactory**

You are here: [ROOT/runtime/transformers](#)

Child list of the current context:

Name	Class name
com.sap.pct.pdk.transformerprovider	java.lang.String - com.sap.pct.pdk.transformerprovider TransformersProvider
MyTransform	java.lang.String - MyTransform TransformersProvider

The links that are displayed after **You are here** indicate the subcontext that you are currently viewing. Use these links to navigate back up the JNDI tree.

1.1.3 Working with the PRT

Purpose

This section describes key considerations when building portal components and services, and includes the following:

- [Permission Model \[Page 26\]](#)
- [Internationalization \[Page 27\]](#)
- [Customizing Caching \[Page 30\]](#)
- [Writing JSP Pages \[Page 33\]](#)
- [Web Services \[Page 33\]](#)
- [Integrating with the SAP J2EE Engine \[Page 33\]](#)

1.1.3.1 Permission Model

Purpose

Access to portal content is controlled via permissions, which are set by the portal administrator.

One mechanism for controlling access to portal components and services in the portal is via security zones, as follows:

1. Developers assign their components and services to a security zone, by specifying the security zone in the `portalapp.xml`.
2. Once the components and services are deployed to the portal, a portal administrator assigns permissions on the security zone. The permissions of a security zone control access to all components and services in that security zone.

The security zone for a component or service is specified by the following `portalapp.xml` properties:

- **Vendor Name** (such as `com.sap`), the default is `UndefinedVendor`
- **Security Area** (such as `NetWeaver.Portal`), the default is `UndefinedSecurityArea`
- **Safety Level** (such as `high_safety`), the default is `UndefinedSafetyLevel`

Portal Runtime

The portal system administrator sets the permission for each security zone, and all components and services in the zone inherit these permissions.



A service's permission only affects access to the service when exposed as a Web service. All components, no matter the user who generated the request, have permission to access all services.

For more information on security zones, see [Security Zones \[External\]](#).

For more information on portal permissions, see [Portal Permissions \[External\]](#)

Setting the Security Zone

The security zone for a component is defined by properties in two places in the deployment descriptor (`portalapp.xml`) file for the application in which the component is deployed:

- **<application-config>**: The `Vendor` and `SecurityArea` properties set the vendor name and security area for all components in the application.

```
<application-config>
  <property name="Vendor" value="sap.com"/>
  <property name="SecurityArea" value="MyCompany"/>
</application-config>
```

- **<component-config>**: The `SafetyLevel` property for each component sets the safety level for that component.

```
<component-config>
  <property name="ClassName" value="com.sap.portal.myComponent"/>
  <property name="SafetyLevel" value="low_safety"/>
</component-config>
```

During deployment, an application's components and services are placed in the appropriate security zone, as defined in the `portalapp.xml`. A portal administrator can then modify the permissions on the security zone or on the component or service.



Content developers and portal administrators should work together in determining what security zones to create and in which security zones to place each component and service.

Changing the Security Zone

To change the security zone for a deployed component or service, change the `Vendor` Name, `SecurityArea` and `SafetyLevel` properties in the `portalapp.xml` file and redeploy the application.

1.1.3.2 Internationalization

Purpose

To enable your portal application to be displayed in different languages, you must create resource bundles that hold all language- or country-specific text strings or other resources.

You can create a different resource bundle for each locale-country combination that you want to support, and then add these resource bundles to your PAR file. Your components can retrieve strings or other resources from these files.

Resource Bundles in PAR Files

The first part of the internationalizing process is resourcing. This involves isolating the locale-specific resources of the source code into modules called resource bundles. Those modules can then be independently added to or removed from the application.

For a portal application, all the locale-specific strings and objects are stored in a set of resource bundles – generally `.properties` files – packaged with the PAR file.

Place resource bundles either in `PORTAL-INF/private/lib` (when packaged in JAR files) or in `PORTAL-INF/private/classes` (as individual files).

Administration Tools

The portal comes with tools that enable content administrators to internationalize strings that they define in an iView, page or workset, such as the name of the iView or the value of iView properties.

Content administrators can also provide translations for text strings within resource bundles, essentially creating new resource bundles for locales that are not supported.

For more information, see [Portal Content Translation \[External\]](#).

1.1.3.2.1 Lookup of Resource Bundle

The resource bundle that is retrieved by a portal component is based on the resource bundle name defined in the deployment descriptor, plus the regional settings for the current user.

The rules for selecting the locale-specific resource bundle are defined by the Java programming language.

The locale is determined by the following properties in the order shown:

1. Component locale

This locale is defined by the following component profile properties:

- `ForcedRequestLanguage`
- `ForcedRequestCountry`

This enables you to force a component to use a specific local, such as for administration components.

2. Portal mandatory locale

This locale is defined in `prtDefault.properties` by the following properties:

- `request.mandatorylanguage`
- `request.mandatorycountry`

This locale is useful for administrators setting up a portal environment.

3. User locale

This locale is defined in the profile of the current user.

4. Request locale

Portal Runtime

The request locale is defined by the browser. This is used, for example, for anonymous users or users that do not have locales defined in their profiles.

5. Portal default locale

This locale is defined in `prtDefault.properties` by the following properties.

- `request.defaultlanguage`
- `request.defaultcountry`

6. System default locale

This is the Java default locale defined by the system, either by the operating system or by the JVM.

For example, if the current default locale is `en_US`, the locale of the component is `fr_CH` and the resource bundle name is `localization`, the portal will look for resource bundles in the following order:

1. `localization_fr_CH`
2. `localization_fr`
3. `localization_en_US`
4. `localization_en`
5. `localization`

The resource bundles must be in the *PORTAL-INF/private/* folder.

1.1.3.2.2 Accessing Strings

Accessing resource strings in a portal component is similar to accessing resource strings from any Java application. Retrieve the resource bundle object (`ResourceBundle`) and then call `getString()` for your string, as shown below:

```
import com.sapportals.portal.prt.component.*;
import com.sapportals.portal.prt.resource.*;
import java.util.ResourceBundle;

public class HelloWorldComponent extends AbstractPortalComponent {

    public void doContent(
        IPortalComponentRequest request,
        IPortalComponentResponse response) {

        ResourceBundle resource = request.getResourceBundle();
        response.write(resource.getString("GREETING"));
    }
}
```

In the example above, the request object retrieves a resource bundle whose name is indicated by the `ResourceBundleName` property in the `<component-config>` element for the current component in the deployment descriptor.

You can retrieve a resource bundle for another component by creating an `IPortalComponentContext` object for the other component, and then getting its resource bundle, as shown in the following example:

```
IPortalComponentContext myPortalComponentContext =
    request.getComponentContext("myApp.myComp");
ResourceBundle resource =
```

```
myPortalComponentContext.getResourceBundle(request.getLocale());
```

1.1.3.2.3 Encoding

Data sent from the browser to a portal component for processing is converted based on the following properties, which are defined in the portal configuration file:

Name	Default Value	Description
<code>runtime.doubleByteSupport</code>	<code>true</code>	Convert data from browser. If <code>runtime.doubleByteSupport</code> is set to <code>false</code> , the encoding is the encoding defined at the servlet container level.
<code>runtime.doubleByteSupport.encoding</code>	<code>UTF-8</code>	Encoding for decoding data from browser and encoding data sent to browser.

Data is read in Unicode format and then manipulated accordingly. The locale of the browser is found with the `getLocale()` method of the original servlet request.

1.1.3.3 Customizing Caching

Purpose

The portal implements a persistent cache for storage of portal content of requests of type `content`. The portal provides the following ways of controlling the cache for a portal component:

- [Component Profile \[Page 30\]](#): You can set profile properties in the deployment descriptor.
- [Caching Interfaces \[Page 31\]](#): You can implement interfaces defined by the `com.sapportals.portal.prt.component` package.

1.1.3.3.1 Component Profile

The following table describes the profile properties that you can define in the deployment descriptor – in the `<component-profile>` element for your component – in order to control the caching of your component's content:

Property	Description
----------	-------------

CachingLevel	<p>Indicates the type of cache, as follows:</p> <ul style="list-style-type: none"> • None (default): The component is not cached. • Shared: The component's content is cached and the cache is shared among all the users. • User: The component's content is cached separately for each user. • Session: The component's content is cached all the time the (servlet) session is running, whether or not a user is connected. This level is used to cache information related to the browser's sessions.
ValidityPeriod	<p>Indicates how long the cache is valid, in milliseconds.</p> <p>-1 indicates the cache never expires.</p>
ALLOW_BROWSER	<p>Indicates if browser caching is used:</p> <ul style="list-style-type: none"> • Yes (default): Browser caching enabled. • No: Browser caching disabled, for cases where you do not want to store cache locally. • BrowserOnly: Only browser caching is used; server stores only cache meta-data.

The following is a deployment descriptor that defines a component with a `CachingLevel` of session and `ValidityPeriod` of 2 seconds:

```
<application>
  <application-config/>
  <components>
    <component name="session">
      <component-config>
        <property name="ClassName" value="myApp.myCompnent"/>
      </component-config>
      <component-profile>
        <property name="CachingLevel" value="Session"/>
        <property name="ValidityPeriod" value="2000"/>
      </component-profile>
    </component>
  </components>
</application>
```

1.1.3.3.2 Caching Interfaces

The `com.sapportals.portal.prt.component` package defines the following interfaces that, if implemented by a component, control the caching of the component's content:

- [ICachablePortalComponent \[Page 32\]](#): Determines the caching level and whether the cache has expired.
- [ICacheValidator \[Page 33\]](#): Determines whether the cache is still valid based on a string that was stored with the cache.
- [ICacheDiscriminator \[Page 33\]](#): Enables the storage of different caches for this component and to label each cache. When called again, the component can specify a label and the cache with that label is returned, if it exists.



A portal administrator can force the use of the cache properties defined in the component profile instead of these caching methods by setting the component profile property `ForceProfileCachingParams` to `true`.

Automatic Caching Expiration

Cached content expires automatically in the following cases:

- A request or a component event is sent to the component.
- The mode is `REFRESH`.
- An exception is raised during the request cycle.

1.1.3.3.2.1 ICachablePortalComponent

The `ICachablePortalComponent` interface defines the following methods for determining the caching level and whether the cache has expired:

- `getCachingLevel`: Sets the caching level, either none, shared, session or user.

```
public CachingLevel getCachingLevel(
    IPortalComponentRequest request)
```

Returns a `CachingLevel` object that represents a caching level. For example, a component can change its caching level at runtime and decide not to be cached, depending on request parameters.

If this method is not implemented, the default implementation from `AbstractPortalComponent` returns the value set in the `CachingLevel` property.

- `hasExpired`: Determines if the cache has expired. `True` indicates the cache has expired.

```
public boolean hasExpired(IPortalComponentRequest request,
    long creationTime,
    long currentTime)
```

If this method is not implemented, the default implementation from `AbstractPortalComponent` returns the value set in the `ValidityPeriod` property.

Example

The following is an example of how to use the `ICachablePortalComponent` interface:

```
import com.sapportals.portal.prt.component.*;

public class MyComponent extends AbstractPortalComponent
    implements ICachablePortalComponent {

    public boolean hasExpired(IPortalComponentRequest request,
        long creation,
        long current) {
        if ((current - creation) > 10000) {
            return true;
        }
        return false;
    }
}
```

```
    }

    public void doContent(IPortalComponentRequest request,
                        IPortalComponentResponse response) {
        long current = java.lang.System.currentTimeMillis();
        response.write("<H2>" + new java.util.Date(current) +
"</H2>");
    }
}
```

1.1.3.3.2.2 ICacheValidator

The `ICacheValidator` interface enables you to save a string value (validation key) with the cache and then, during the next request, determine whether the cache is still valid based on the string that was stored.

The interface defines the following methods:

- `isCacheValid`: Determines whether the cache is still valid. The key that was stored with the cache is passed to the method to help determine the validity of the cache.

```
public boolean isCacheValid(IPortalComponentRequest request,
                        String key);
```

The key parameter is the string that was returned from the `getValidationKey()` method during the last request.

- `getValidationKey`: Returns a key to be passed to the `isCacheValid()` method on the next request.

```
public String getValidationKey(IPortalComponentRequest request);
```

Example

The following code displays an applet, whose size is read from the component profile. The `ICacheValidator` interface is used to invalidate the cache when these values change:

```

public class DisplayAppletComponent
    implements ICacheValidator, ICachablePortalComponent {

    public void doContent(
        IPortalComponentRequest request,
        IPortalComponentResponse response) {

        // Display applet of a certain size
        int width = getAppletWidthFromProfile(request);
        int height = getAppletHeightFromProfile(request);
        displayApplet(width, height);
    }

    public CachingLevel getCachingLevel() {
        return CachingLevel.USER;
    }

    /**
     * The method determines if the key has changed. If it has,
     * the cache is invalidated.
     */
    public boolean isCacheValid(IPortalComponentRequest request,
                               String key) {
        int width = getAppletWidthFromProfile(request);
        int height = getAppletHeightFromProfile(request);
        String newKey = computeKey(width, height);
        return key.equals(newKey);
    }

    /**
     * Saves with the cache a key based on profile properties
     */
    public String getValidationKey(IPortalComponentRequest request)
    {
        int width = getAppletWidthFromProfile(request);
        int height = getAppletHeightFromProfile(request);
        // Computes a key with the width and height
        String key = computeKey(width, height);
        return key;
    }

    private String computeKey(int width, int height) {
        String key = "" + width + "|" + height;
        return key;
    }
}

```

1.1.3.4

1.1.3.4.1.1 ICacheDiscriminator

The `ICacheDiscriminator` interface enables you to save different caches for a component, each with a different label, or cache discriminator. When another request is made for the component, the portal requests the cache discriminator from the component and then retrieves cached content if one of the caches has the specified cache discriminator.

Portal Runtime

The interface defines the following method:

- **getCacheDiscriminator:** Returns a string for labeling the component's content when placed into the cache.

```
public String getCacheDiscriminator(IPortalComponentRequest request)
```

Example

A component displaying weather reports can cache content for each city.

```
package com.sapportals.portal.prt.test.component;
import com.sapportals.portal.prt.component.*;
import com.sapportals.portal.prt.event.IPortalRequestEvent;

public class WeatherComponent implements ICacheDiscriminator {
    private final String SELECTED_CITY = "SelectedCity";
    private final String CITY_PROPERTY = "City";

    public void doContent(IPortalComponentRequest request,
                        IPortalComponentResponse response) {
        String cityName =
            (String) request.getNode().getValue(CITY_PROPERTY);

        // Add content here
    }

    /**
     * Stores the selected city in the component profile
     */
    public void doSelect(IPortalComponentRequest aRequest,
                        IPortalRequestEvent event) {
        String cityName =
            (String) aRequest.getParameter("SELECTED_CITY");
        aRequest.getNode().putValue(CITY_PROPERTY, cityName);
    }

    /**
     * Uses the city name as the cache discriminator.
     */
    public String getCacheDiscriminator(
        IPortalComponentRequest request) {
        String cityName =
            (String) request.getNode().getValue(CITY_PROPERTY);
        return cityName;
    }
}
```

1.1.3.4.2 Browser Caching

The portal supports browser caching using HTTP conditional requests. Enable browser caching by setting `ALLOW_BROWSER` property in the component's profile.

The following are the steps that occur if browser caching is enabled:

1. If browser caching is enabled, an HTTP header `LastModified` is sent to the browser with the last-modified time.

Portal Runtime

2. After receiving a response with a `LastModified` header, the browser sends an `If-Modified-Since` header to the server if the same request is made again. The header contains the time from the `LastModified` header.
3. The `If-Modified-Since` value can be compared to the cached content timestamp of the root node.

If they match, the HTTP response header `NOT_MODIFIED` is sent to the browser.

Otherwise, the content is regenerated, stored in the cache, and sent again to the browser.

A portal component can use browser caching if:

- The root component node is cached.
- All cached nodes in the Portal Object Model (POM) have the property `ALLOW_BROWSER` set to `Yes` or `BrowserOnly` in their profile, as shown below:

```
<component name="user2">
  <component-config>
    <property name="ClassName" value="myApp.myComponent"/>
  </component-config>
  <component-profile>
    <property name="CachingLevel" value="User"/>
    <property name="ALLOW_BROWSER" value="BrowserOnly"/>
  </component-profile>
</component>
```

1.1.3.5 Writing JSP Pages

Purpose

The Portal Runtime enables you to run JSP pages in the portal, and to write portal components as JSP pages.



The portal has its own implementation of JSP, so not all previously written JSP pages can be run in the portal. For more information on differences in the portal implementation of JSP, see *JSP Objects and Directives*.

The following shows the JSP versions used by the Portal Runtime and the SAP J2EE engine for different versions of the portal:

Portal Version	Portal Runtime	SAP J2EE Engine
SP2	J2EE 1.2 (JSP 1.1)	J2EE 1.2 (JSP 1.1)
NetWeaver 04	J2EE 1.2 (JSP 1.1)	J2EE 1.3 (JSP 1.2)

All JSP files must be placed in the private area (`PORTAL-INF` folder) of the PAR in which they are packaged.

When the JSP is first accessed, the portal creates a `.java` file that defines a class that extends `AbstractPortalComponent` that is based on the JSP page, and then compiles the file into a `.class` file. When the JSP is called, it is this class that is executed, not the JSP. The name of the java file is `_sapportalsjsp_[name of JSP file].java`.

Portal Runtime

The package name of the new portal component class is based on the folder in which the JSP is placed. For example, for a JSP page whose path is

`PORTAL-INF/jsp/select/customer/default/entry.jsp`, the java class `_sapportalsjsp_myJSP` in package `jsp.select.customer.default` is created in the file `_sapportalsjsp_myJSP.java`.

The portal places the .java and .class files in a new folder called `work` in the `/WEB-INF/portal/portalapps/<application name>` folder, where `application name` is the portal application in which the JSP is packaged.

Development Vs. Production Mode

In production mode, a change in the JSP page on the file system of the portal does not cause the JSP to be recompiled. For performance reasons, the page is recompiled, if necessary, only when the application in which the page is packaged is redeployed.

During development and testing, it may be easier to make changes to a JSP on the file system than to make changes in NetWeaver Developer Studio and then redeploy the application. Therefore, in development mode, the JSP page is checked to see if it was modified each time the page is called.

Encoding

When a portal application uses a JSP resource that contains double-byte characters, the portal needs to know the character set used to create the file. The portal can then parse the file in the correct encoding and generate the corresponding output in UTF-8 encoding (or in the encoding specified in the `runtime.doubleByteSupport.encoding` property in the `prtcentral.properties` configuration file).

Specify the encoding in the `pageEncoding` attribute of the page directive in the JSP file:

```
<%@page pageEncoding="shift_JIS" %>
```

1.1.3.5.1 Packaging JSP Pages

JSP pages can be packaged as a portal component, or as a resource contained in a portal application.

- **Portal Component:** You can define a portal component, and provide as the implementation a JSP page instead of a Java class that extends `AbstractPortalComponent`, as described in [JSP as Portal Component \[Page 33\]](#).

You define the portal component in the application's deployment descriptor, and specify that the portal run the JSP page whenever the component is called.

- **Standalone Resource:** You can call and execute a JSP page from within a portal component, as described in [JSP as Standalone Resource \[Page 33\]](#). The output of the JSP page is included in the output for the component.

1.1.3.5.1.1 JSP as Portal Component

To create a portal component from a JSP page, create a `<component>` element in the deployment descriptor of your application.

Specify the following properties in the `<component-config>` element:

Property	Value
----------	-------

Portal Runtime

JSP	The path to the JSP file that is run when the component is called. The path is relative to the <code>PORTAL-INF</code> folder of your PAR.
ComponentType	jspnative

You do not have to create a `className` property.

1.1.3.5.1.2 JSP as Standalone Resource

To execute a JSP file within a component and include the output in the output of a component, specify the JSP page in the `include()` method of the portal request object.

The following shows how to include the output from a JSP page contained in the same application as the calling component:

```
public class CheckBoxComponent extends AbstractPortalComponent {
    public void doContent(
        IPortalComponentRequest request,
        IPortalComponentResponse response) {

        IResource jspResource =
            request.getResource(IResource.JSP, "jsp/checkres.jsp");
        response.include(request, jspResource);
    }
}
```

Above, the JSP page `checkres.jsp` in the directory `/PORTAL-INF/jsp` is executed.

The following shows how to include a JSP page that is packaged in a different application:

```
public class CheckBoxComponent extends AbstractPortalComponent {
    public void doContent(
        IPortalComponentRequest request,
        IPortalComponentResponse response) {

        IResource jspResource =
            request.getResource("JSPValidation", IResource.JSP,
                "/jsp/include/checkres.jsp");
        response.include(request, jspResource);
    }
}
```

Above, the JSP page `checkres.jsp` in the directory `/PORTAL-INF/jsp/include` of the application `JSPValidation` is executed.

1.1.3.5.2 JSP Objects and Directives

You can access the standard JSP objects from within your page. However, some of the objects are portal implementations and not the standard Java implementations, and therefore provide different functionality.

Objects

The following table shows the JSP objects accessible from within a JSP page:

Object	Class	Description
<code>request</code>	<code>javax.servlet.ServletRequest</code>	Standard request object
<code>response</code>	<code>com.sapportals.portal.prt.</code>	Portal response object

Portal Runtime

	IPortalComponentResponse	
pageContext	javax.servlet.jsp.PageContext	Standard page context for this JSP page. <i>(The forward functionality is not supported.)</i>
componentRequest	com.sapportals.portal.prt.IPortalComponentRequest	Portal component request object
session	javax.servlet.http.HttpSession	Standard session object
application	javax.servlet.ServletContext	Standard servlet context
out	javax.servlet.jsp.JspWriter	Standard object for writing to output stream
config	javax.servlet.ServletConfig	Standard object for writing to output stream
page	com.sapportals.portal.prt.IPortalComponentContext	Standard servlet configuration object
exception	java.lang.Throwable	Standard exception object, for use by error pages to view the error that caused the page to be called.

Directives

The following directives have special implementations:

- [Page \[Page 33\]](#)
- [Include \[Page 33\]](#)

1.1.3.5.2.1 Page Directive (ErrorPage Attribute)

The `errorPage` attribute of the `page` directive enables you to handle errors by calling a portal component as well as another Web resource, such as a JSP or HTML page.

The following is an example of a page directive:

```
<%@ page errorPage="prt:component:MyComponent" %>
```

The following are the types of values that are valid for the `errorPage` attribute:

Type of Resource	Syntax
JSP or HTML page in the current portal component	<code>error.html</code> or <code>error/myError.html</code> The value is a path, relative to the JSP page.
JSP or HTML page in another portal component	<code>prt:componentres:MyApp.MyComp,error/error.jsp</code>
Portal component	<code>prt:component:MyApp.MyComp</code> <code>MyApp.MyComp</code> is a full-qualified name of a component.

1.1.3.5.2 Include Directive

The `include` directive enables you to add the content produced by a portal component, in addition to another JSP page or a static page.

Directive	Description
<code><jsp:include page="[path]" /></code>	The file is considered static content and is included in the output, as is. Use this directive for HTML and other static content.
<code><%@ include file="[path]" %></code>	The file is considered as JSP code and is parsed and executed before it is added to the output.

The following values are valid for the `page` or `file` attribute:

Type of Resource	Syntax
JSP or HTML page in the current portal component	<code>myInclude.html</code> or <code>myIncludes/myInclude.html</code> The value is a path, relative to the JSP page.
JSP or HTML page in another portal component	<code>prt:componentres:MyApp.MyComp,</code> <code>myIncludes/myInclude.html</code>
Portal component (for page attribute only)	<code>prt:component:MyApp.MyComp</code> <code>MyApp.MyComp</code> is a full-qualified name of a component.

1.1.3.5.3 Tag Libs

JSP pages in the portal can use custom tag libs by doing the following:

- Package the class files that implement the tag lib in the `lib` or `classes` directory of the PAR file's `PORTAL-INF` folder.
- Add a directive to any JSP file that references the tag lib's `.tld` file, as follows:

```
<%@ taglib uri="<uri>" prefix="[prefixname]" %>
```

The following shows the format for the `uri` attribute, above:

URI	Description
Path of <code>.tld</code> file	The path to the tag lib's <code>.tld</code> file, relative to the JSP file.
Alias to <code>.tld</code> file, in the following format: <code>prt:taglib:<alias></code>	An alias for the path to the <code>.tld</code> file is defined by a <code><property></code> element in the <code><component-profile></code> element. For example, an alias for the portal's HTMLB tag lib can be created by adding a property called <code>tlhtmlb</code> and set the property to <code>/SERVICE/com.sap.portal.htmlb/taglib/htmlb.tld</code> . The directive's URI can then be set to <code>prt:taglib:tlhtmlb</code> .

1.1.3.5.3.1 Built-In Tag Libs

All JSPs in the portal that are compiled in a portal component can use built-in portal tag libs by doing the following:

- Add a directive to any JSP file that references the tag lib. The following shows the directives for the HTMLB and Layout tag libs:

```
<%@ taglib uri="/SERVICE/com.sap.portal.htmlb/taglib/htmlb.tld"
prefix="hbj" %>
```

```
<%@ taglib
uri="/SERVICE/com.sap.portal.pagebuilder/taglib/layout.tld"
prefix="lyt" %>
```

- Add a `ServicesReference` reference to the tag lib service. The following shows the reference for the HTMLB and Layout tag libs:

```
<property name="ServicesReference" value="com.sap.portal.htmlb"/>
```

```
<property name="ServicesReference"
value="com.sap.portal.pagebuilder"/>
```

1.1.3.5.4 Beans

JSP pages in the portal can reference Java beans as in standard JSP pages.

In the portal the **scope** attribute of the `<jsp:bean>` action, which is used to instantiate a bean, acts somewhat differently, as shown in the following table, which lists the values that are valid for the **scope** attribute:

Scope	Description	How to Reference Bean from Component
page	Bean is accessible within the current page.	<code>pageContext</code> object
request	Bean is accessible within the current request.	<code>request</code> object
session	Bean is accessible within the current user session.	<code>PortalComponentSession</code> object, from <code>request.getSession()</code>
application	Bean is accessible from pages that process requests that are in the same application as the one in which they were created.	<code>PortalComponentContext</code> object All references to the object are released when the the <code>PortalComponentContext</code> object is reclaimed.

1.1.3.5.5 Servlets

The Portal Runtime enables you to run servlets and package them as portal components.

Procedure

Portal Runtime

1. Create a servlet, that is, a java class that extends `HttpServlet`.
2. In the deployment descriptor of your portal application, define a configuration property named `ComponentType` and assign it a value of `servlet`, as shown below:

```
<components>
  <component name="ServletTest">
    <component-config>
      <property name="ClassName"
value="com.sapportals.portal.prt.component.ServletTest"/>
      <property name="ComponentType" value="servlet"/>
    </component-config>
  </component>
```

3. Put the compiled servlet class in the private `lib` or `classes` folder.

1.1.3.6 Web Services

The Web AS engine provides tools for creating Web services from existing components, such as enterprise Java beans or Java classes.

- To create a Web service that exposes a portal service, see [Exposing Portal Services as Web Services \[Page 33\]](#).
- To call a Web service from a portal service or application, see [Calling a Web Service \[Page 33\]](#).

1.1.3.6.1 Exposing Portal Services as Web Services

To expose a portal service as a Web service, use the Web Service Creation Wizard, which is part of the NetWeaver Developer Studio.

Prerequisites

- The portal service must be created as part of a standalone portal application packed in a development component (DC).
When creating the DC for the portal application that will contain the portal service, select *Portal Application Standalone* in the new development component wizard.
- The portal interface on which the portal service and Web service are based must be defined in the same project as the portal service.

Procedure

1. In NetWeaver Developer Studio, open the Web Services perspective.
2. Right-click the interface that you want to expose, and select **New → Web Service**.

The Web Service Creation Wizard is displayed. For information on running the wizard, see [Web Service Creation Wizard \[External\]](#).



The wizard requires you to specify whether the new Web service will call the current portal service using an existing alias, or whether you want to create a new alias for the service.

It is recommended that you create a new alias. This creates a new portal service based on the same interface. This enables you to set different configuration parameters, including security parameters, for the service when it is called from within the Web AS and when called via a Web service.

3. Build the project, which creates an `.sda` file.
4. Deploy the `.sda` file via the Software Deployment Manager (SDM).

Results

The Web Service Creation Wizard creates the following files:

- **<Web Service Name>Wsd.wsdef**
- **<Web Service Name>Vi.videf**
- **ws-deployment-descriptor.xml**

When the project is built into an `.SDA` file and deployed, the Web AS creates a Web service based on these files. For more information on these files, see [Creating a Web Service \[External\]](#).

When a Web service call is received, the Web AS dispatches the call to a Portal Runtime application (Portal Runtime Web Service Container), which makes the required portal service call and returns the response.

Checking Deployment

The new Web service is deployed to the Web AS. To check the deployment, open the Web Services Navigator to view all the Web services deployed to the Web AS.

The Web Services Navigator is located at `http://<server>:<port>/wsnavigator`.

1.1.3.6.2 Calling a Web Service

To call a Web service from a portal component or service, you need to create a J2EE deployable proxy for the Web service, and then call the proxy.

For information on creating a deployable proxy for a Web service, see [Consuming a Web Service \[External\]](#).

For more information on calling a J2EE application, see [Calling J2EE Applications from Portal Applications \[Page 33\]](#).

1.1.3.7 Integrating with the SAP J2EE Engine

Purpose

This section describes how to integrate portal components and services with the SAP J2EE Engine, and describes the following tasks:

- [Calling Portal Applications from J2EE Applications \[Page 33\]](#)
- [Calling J2EE Applications from Portal Applications \[Page 33\]](#)
- [Packaging PARs in J2EE Applications \[Page 33\]](#)

1.1.3.7.1 Calling Portal Applications from J2EE Applications

This section describes how J2EE applications can access portal components and portal services.

Procedure

1. Specify the `reference` tag.

The `reference` tag is specified in the `application-j2ee-engine.xml`. It has the following specifics:

- Attribute: `reference-type`.
 - Attribute value: `hard` or `soft`.

References to portal applications are treated in a unique way. This means there is no difference if *hard* or *weak* reference type is used.

2. Specify the `reference-target` tag.

- Tag value: The name of the portal application or an alias defined for it:
 - Attribute: `target-type`. Attribute value: `application`.

The value `application` identifies the reference target as the portal application:

- Attribute: `provider-name`. Attribute value: `sap.com`.

The value of this attribute should correspond to the property *Vendor* of the `portalapp.xml`. By default, if no *Vendor* is provided in the portal application descriptor of the referenced application, the `provider-name` in the `application-j2ee-engine.xml` is `sap.com`.

```
<reference-target target-type="application" provider-name="sap.com">  
com.sap.portal.core.examples.PARInEARPortalApp  
</reference-target>
```

3. Access the application.

The application is accessed via JNDI. The lookup is made to the portal registry. There is currently no JNDI schema defined for the portal registry. Therefore, the object factory for the portal registry must be specified in the environment of JNDI through:

- the environment variable: `Context.INITIAL_CONTEXT_FACTORY`. It must have the value:
`com.sapportals.portal.prt.registry.PortalRegistryFactory`.
- The lookup is made using the following JNDI path:
`/broker/services/<service name>`, and you can use one of the following notations:
 - `<service name>` is the key of the service as defined in its interface.
 - `<application name>.<service name>` can be used as defined in the `portalapp.xml` file of the application which provides the service.

Example

The following is an example of a WebDynpro application that uses the portal transformation service to transform an XML file into HTML. The XML file is in RSS format, a common format for providing news on the Internet. The example shows how to read the current news from Yahoo and transform the XML to HTML using an XSL style sheet located in `c:\temp`.

```
import
com.sapportals.portal.prt.service.xsltransform.IXSLTransformService;
import
com.sapportals.portal.prt.service.xsltransform.IWPXSLTransformer;

public class HelloWorld {
    public void wdDoInit() {
        String output;
        String line;
        String rssStyleSheetFileName = "c:/temp/rsshtml20.xsl";
        String topStoriesURLName =
            "http://rss.news.yahoo.com/rss/topstories";
        URL rssStyleSheetURL = null;
        URL topStoriesURL = null;
        StringBuffer news = new StringBuffer();
        StringBuffer styleSheet = new StringBuffer();

        Hashtable env = new Hashtable();

        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sapportals.portal.prt.registry.PortalRegistryFactory");

        InitialContext context = new InitialContext(env);

        IXSLTransformService xslTransformService =
            (IXSLTransformService) context.lookup(
                "/broker/services/" + IXSLTransformService.KEY);

        rssStyleSheetURL = new URL("file://localhost/" +
            rssStyleSheetFileName);

        InputStreamReader iReader =
            new InputStreamReader(rssStyleSheetURL.openStream());
        BufferedReader bReader = new BufferedReader(iReader);

        while ((line = bReader.readLine()) != null) {
            styleSheet.append(line);
        }

        IWPXSLTransformer rssTransformer =
            xslTransformService.getTransformer(
                new StringReader(styleSheet.toString()));

        topStoriesURL = new URL(topStoriesURLName);

        iReader = new InputStreamReader(topStoriesURL.openStream());
        bReader = new BufferedReader(iReader);

        while ((line = bReader.readLine()) != null) {
            news.append(line);
        }
    }
}
```

```

        output = rssTransformer.transform(news.toString());
    }
}

```

1.1.3.7.1.1 Accessing the PCD from a J2EE Application

J2EE applications can access content stored in the Portal Content Directory (PCD) using JNDI.

In the following example, a Web Dynpro application accesses an object in the PCD. The content of the PCD is protected against impermissible access. Therefore, the SAP UME user must be put in the JNDI environment to identify the current user. In the example below this user is obtained from the Web Dynpro specific user.

The example shows the lookup of the data only. The `InitialContext` is a `javax.naming` class.

```

import com.sap.tc.webdynpro.services.sal.um.api.WDClientUser;
import com.sap.security.api.IUser;

// class implementation

    // method implementation

        // getting the SAP user.
        IUser currentUser =
WDClientUser.getCurrentUser().getSAPUser();

        Hashtable env = new Hashtable();
        env.put(Context.SECURITY_PRINCIPAL, currentUser);

        try {
            InitialContext context = new InitialContext(env);

            Object obj = context.lookup(
                "pcd:/com.sap.portal.system/applications/
                com.sap.portal.ivs.global/services/producer");

        } catch (NamingException e) {
            // handle the exception
        }

```

1.1.3.7.2 Calling J2EE Applications from Portal Applications

Portal applications can access J2EE applications, such as EJBs and servlets, as follows:

1. Specify the reference.

The reference is defined in `portalapp.xml` either in `PrivateSharingReference` or `PublicSharingReference` using the J2EE application prefix `SAPJ2EE::`.

```

<property name="PrivateSharingReference"
value="SAPJ2EE::sap.com/Hello"/>

```

2. Access the application.

The application is accessed via JNDI. The following is an example of a portal component accessing an EJB.

```
public void doContent(
    IPortalComponentRequest request,
    IPortalComponentResponse response) {
    String key = "Hello/Stateless/HelloStatelessBean";

    try {
        Context context = new InitialContext();

        Object obj = context.lookup(key);
        HelloHome home =
            (HelloHome) P4ObjectBroker.init().narrow(obj,
HelloHome.class);
        Hello hello = home.create();
        response.write("Bean message: " + hello.getMessage());
    }
}
```

Referencing Elements

You can reference an application, connector, library, interface or service by specifying the the J2EE object in the `portalapp.xml`, either in the `SharingReference` or `PrivateSharingReference` property. Whether you should use `SharingReference` or `PrivateSharingReference` depends on your purpose:

- If the referenced application, library or service has to be used in the API of the portal application, use the `SharingReference`.
- If the reference is for the core of the portal application, use `PrivateSharingReference`.

The format of the J2EE object in the `SharingReference` or `PrivateSharingReference` property is as follows:

- **Application:** `SAPJ2EE::<provider prefix/application name>`
- **Connector:** `SAPJ2EE::<provider prefix/connector name>`
- **Service:** `SAPJ2EE::service:<service name>`
- **Library:** `SAPJ2EE::library:<library name>`
- **Interface:** `SAPJ2EE::interface::<interface name>`



The format includes the provider prefix for an application or connector, but not for services, libraries and interfaces. The provider prefix is separated from the application name by slash (/).

For more information on sharing references, see [Application Configuration \[Page 9\]](#).

Example

This is an example of a portal application referencing the J2EE application `sap.com/Hello`, the J2EE Engine service `P4` and the library `TestLibrary`. The references are specified in `PrivateSharingReference`. This means that the references are established between the core of the portal application and the referenced application, service and library.

```
<application>
  <application-config>
    <property name="SharingReference" value="" />
  </application-config>
</application>
```


Portal Runtime

```

        <property name="PrivateSharingReference"
            value="SAPJ2EE::sap.com/Hello,
SAPJ2EE::service:p4,SAPJ2EE::library:TestLibrary"/>
    </application-config>
</application>

```

1.1.3.7.3 Packaging PARs in J2EE Applications

A PAR file can be packaged in an Enterprise Application Archive (EAR) file. When the EAR file is deployed using J2EE deployment facilities, the PAR is forwarded to the PRT container.

When a PAR file is deployed in EAR file, it cannot later be deployed with the standard deployment tools. It should always be deployed in an EAR file.

Procedure

To use the portal application through a J2EE Application, the portal application must be started as a reference. List the module in the `application-j2ee-engine.xml` deployment descriptor of the J2EE application in the *deployment reference*. The reference contains the following tags:

Tag	Value
modules-additional	<none>
module	<none>
entry-name	<i><the name of the portal application including the extension></i>
container-name	PortalRuntimeContainer

Cluster Considerations

- **When Deploying:** If a PAR is deployed as part of an EAR in a cluster, then only the J2EE node in which the deployment was triggered will receive the PAR file in its PRT container. This container then uploads the PAR in the PCD and triggers the update of the local deployment of all portal nodes. The PRT containers on all other nodes will only be notified about the deployment.
- **When Undeploying:** The PRT container on the node on which undeploy was triggered will remove the PAR from the PCD and notify all portal nodes to remove the local deployment. The PRT containers on the other nodes do nothing.

Multiple File Deployment

If multiple PARs are contained in one EAR, the PARs are uploaded in one step to the PCD. In addition, the update of the local deployment on all portal nodes is made in one step too.

If two EAR files contain one and the same PAR file, then the first PAR file will be removed and replaced by the second PAR file. A warning message is displayed in the portal log files.

Example

This is the part of a deployment descriptor for an EAR file containing two PAR files:

```

<modules-additional>
  <module>
    <entry-name>

```

```

        com.sap.portal.core.examples.PARInEARPortalApp.par
    </entry-name>
    <container-type>
        PortalRuntimeContainer
    </container-type>
</module>
<module>
    <entry-name>
        test.portal.MyPortalTest.par
    </entry-name>
    <container-type>
        PortalRuntimeContainer
    </container-type>
</module>
</modules-additional>

```

1.1.4 Deployment of Applications

Once you create a portal application and bundle it into a PAR file, it must be deployed to a portal, which can be done in the following ways:

- **Archive Uploader Tool:** A single PAR can be uploaded by the Archive Uploader, which is available in the portal at *System Administrator* → *Support* → *Portal Runtime* → *Administration Console*.

This method is used by administrators and developers.

For more information, see [Deploying Applications Via the Archive Uploader \[Page 33\]](#).

- **Eclipse Plug-in:** A single PAR can be uploaded from the NetWeaver Developer Studio directly into a portal via the PAR Export feature of the portal plug-in for Eclipse.

This method is used by developers.

For more information, see [Managing PAR and JAR Files in the Project \[Page 33\]](#).

- **Business Package:** A collection of PARs and portal content can be packaged together and deployed via the SAP Deployment Manager.

This method is generally used by administrators.

For more information, see [Business Package Administration \[External\]](#).

- **EAR File:** PAR files can be included in a standard J2EE EAR file, and the EAR file can be deployed to the SAP J2EE Engine.

This method is used by administrators and developers.

For more information on packaging a PAR in an EAR file, see [Packaging PARs in J2EE Applications \[Page 33\]](#).

For more information on deploying EAR files, see [Deploying EARs \[External\]](#), which is located in the NetWeaver documentation at *Application Platform* → *Java Technology in SAP Web Application Server* → *Development Manual* → *Developing Web Applications* → *Developing J2EE Applications* → *Creating Enterprise Application Projects*.

No matter what method is used to deploy an application, the application is placed into the application repository and a copy is placed on each node in the cluster. For more information on the application repository, see [Application Repository \[Page 33\]](#).

The actions that occur during deployment are described in [Deployment Flow \[Page 33\]](#).

1.1.4.1 Application Repository

All Portal Runtime applications deployed in the portal are stored in the PCD via the application repository, which provides a special JNDI interface to the PCD. There is one application repository per cluster, and all nodes share the same content.

The application repository provides information on all available applications, components and services, as well as application configuration information.

The application repository provides the following JNDI contexts for the PCD:

Context	Description
com.sap.portal.system	Root context
com.sap.portal.system/archives	Lists PAR files deployed.
com.sap.portal.system/applications	Lists applications loaded in the repository.
com.sap.portal.system/configuration	Lists the configuration for the applications.

Local Deployment

Immediately after an application is uploaded to the application repository, a copy is deployed locally to each node in order to improve performance by providing fast access to resources and avoiding extra calls to the repository.



The applications on each node are, in essence, only cached copies of the applications that are stored in the repository. The copies on each node should not be modified directly.

1.1.4.2 Deployment Flow

When an application is deployed to the portal, the following occurs:

- The application is deployed to the Application Repository, as described in *Deployment into Application Repository*.
- A copy of the application is written to each server node in the cluster, as described in *Deployment to Local Servers*.

Deployment into Application Repository

The following describes the steps the PRT takes when you deploy an application:

- The application is placed in the PCD at `par:/archives/<application_name>`.
- The deployment descriptor is extracted and placed in the PCD at `par:/application/<application_name>`.
- Entries for the security zones defined in the application are created, if necessary, in the PCD at `par:/security/<vendor>/<security area>/<safety level>`.
- Any resource bundles are stored at `par:/resource_bundles/<application_name>`.

- The servers in the cluster are notified that a new application has been deployed.

Deployment to Local Servers

After an application is deployed to the Application Repository, all servers in the cluster are notified. The following lists the steps in deploying a copy of the application to each server:

- If a previous version of the application is loaded, this instance is dropped. All instances of applications, components and services that depend on the new application are also dropped.



Applications can be marked as non-releasable in their deployment descriptor, but this only protects against dropping instances of the application's components and services in case the VM runs low on memory.

No matter the setting of this flag, the components and services are dropped when the application is redeployed.

- The application is unzipped and copied to the local server.
- All portal applications that were dropped are restarted.
- All services in the application that are marked to be started when the portal is started are restarted.

How Objects are Instantiated

1. When a component or a service of a portal application is accessed, The object broker checks whether the object is already available and returns it, if available.
2. If the object is not available, the broker tries to get it from the local deployment. The broker checks the revision number of the application to make sure the local deployment is up to date, and then loads the class and instantiates the object, if the local version is up to date.
3. If the local version is not up to date, the new version in the application repository is deployed on the local file system, and the object broker then loads the class and instantiates the object.

The portal application runs and remains in the memory unless a new version is deployed or an administrator decides to release the application. The system can also discard portal application instances when the Java VM needs to free up memory.

1.1.5 Deployment Policy (Enterprise Portal 5.0)

The `DeploymentPolicy` property of a component's configuration determines whether the local deployment is deleted during deployment. The property's value can be one of the following:

- `5.0`: The deployment is backward compatible with Enterprise Portal 5.0. The old folder is not removed and the deployment process simply adds the new content to the folder.
- `Blank`: A standard deployment is performed. The old deployment folder is removed and replaced by the a new one with the new content.

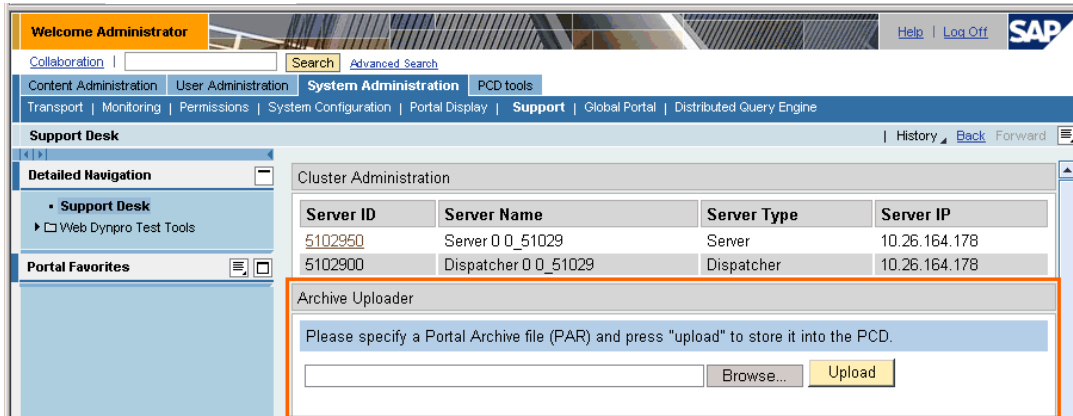
The deployment assumes that the application stores configuration in the application repository, and not locally.

1.1.5.1 Deploying Applications Via the Archive Uploader

Administrators who deploy portal applications use the Archive Uploader, which enables them to upload a PAR file to the PCD. The application is automatically copied to each server node in the cluster.

Procedure

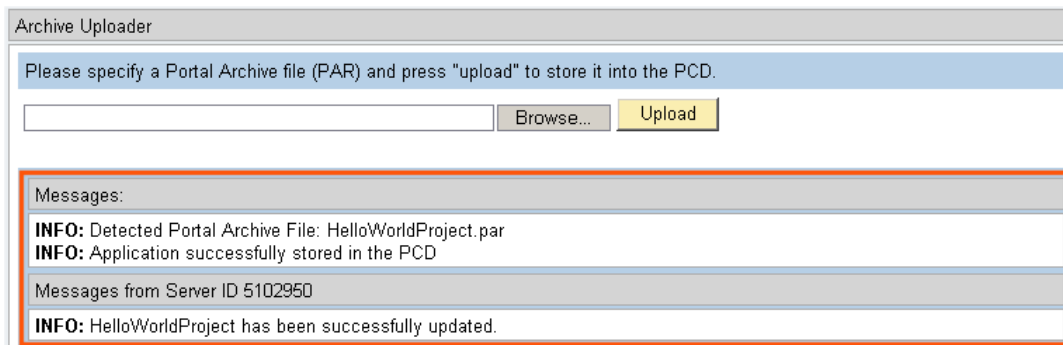
1. Run the Administration Console by clicking *System Administration* → *Support* → *Portal Runtime* → *Administration Console*. The following is displayed:



2. Select the PAR file that you want to upload by clicking *Browse* and selecting the PAR file.
3. Click *Upload*.

Result

The PAR file is uploaded, and status messages are displayed, indicating whether the application was stored in the PCD and copied to each server node in the cluster.



1.1.5.2 Checking Deployment

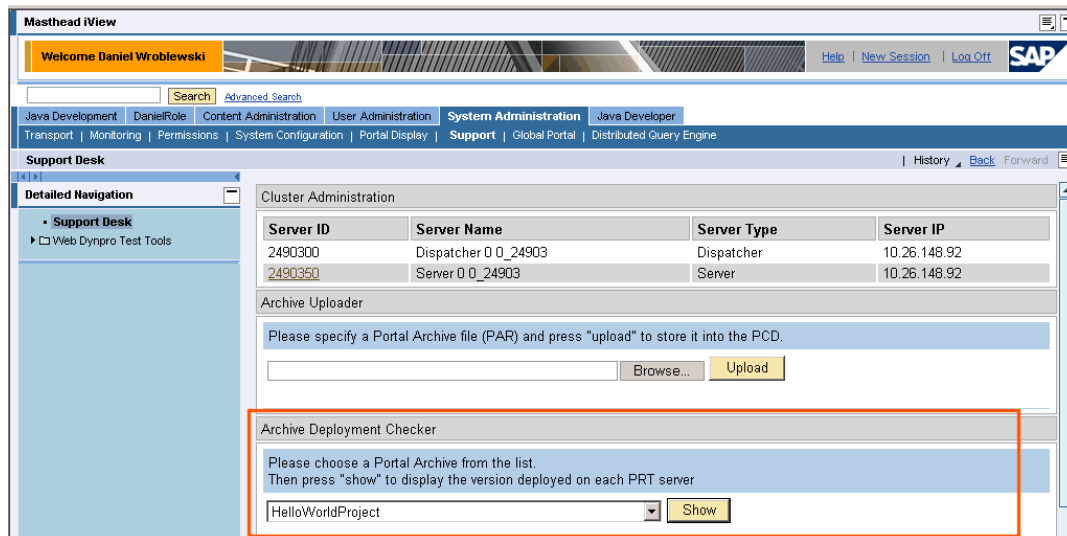
You can use the Administrative Console to check the deployment of a single application or to check all applications deployed on a node.

For more information on how to deploy and remove portal runtime applications, see [Deploying Applications Via the Archive Uploader \[Page 33\]](#).

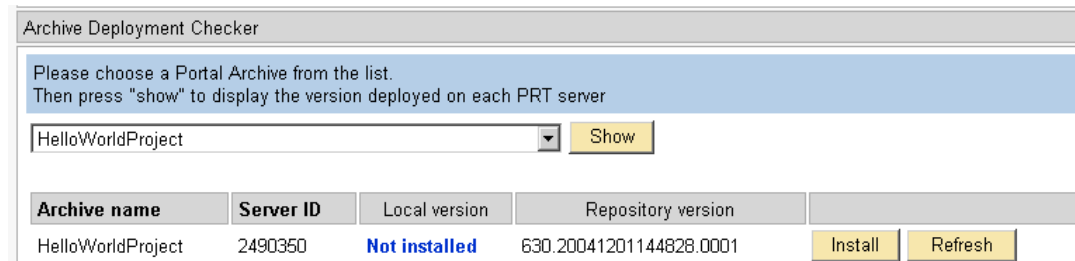
Single Application

The following describes how to check the deployment of a specific application:

1. Run the Administration Console by clicking *System Administration* → *Support* → *Portal Runtime* → *Administration Console*. The following is displayed:



2. Under Archive Deployment Checker, select the application that you want to check, and click *Show*. The following is displayed:



The console shows a line for each server in the cluster, and shows the version of the application in the Application Repository and the version deployed on each server. If the local version is shown in red, the local version is different from the Application Repository version.

The following buttons are provided for each server:

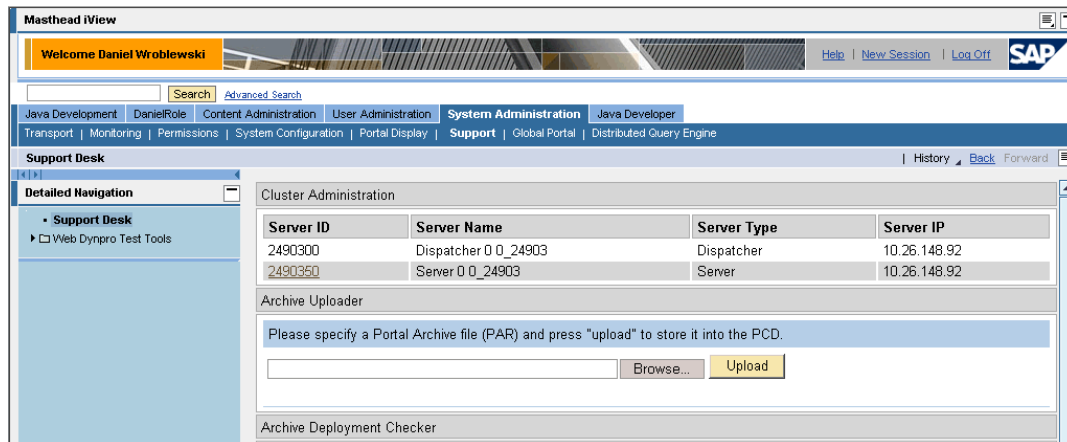
- **Update:** Redeploys to the server the application from the Application Repository.
- **Delete:** Deletes the application from the server. The application remains in the Application Repository and is redeployed automatically when a request for one of its components or services is made. The application can also be redeployed manually by clicking *Install*.
- **Install:** Redeploys the application to the server.

If the application has already been deployed locally, *Update* and *Delete* are displayed. If not, *Install* is displayed and *Not Installed* is displayed for the local version number.

All Applications

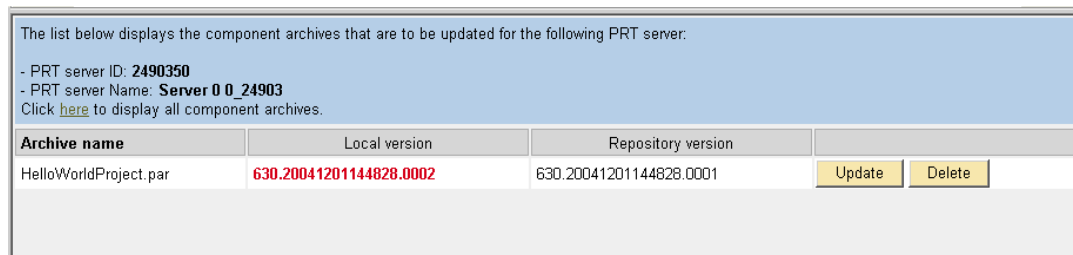
With the Application Console, you can also check the deployment of all applications at once.

1. Run the Administration Console by clicking *System Administration* → *Support* → *Portal Runtime* → *Administration Console*. The following is displayed:



At the top is a list of servers within the cluster.

2. Click the link for the server that you want to check, for example, *2490350* in the above example. The console lists applications whose version in the Application Repository is different than the one deployed on the selected server, as in the following:



In the above example, the console shows the application's version number in the Application Repository (.0001) and the version deployed locally (.0002).

To see all applications in the Application Repository, click *here*.

The following buttons are provided for each application:

- **Update:** Redeploys to the server the application from the Application Repository to the server.
- **Delete:** Deletes the application from the server. The application remains in the Application Repository and is redeployed automatically when a request for one of its components or services is made. The application can also be redeployed manually by clicking *Install*.
- **Install:** Redeploys the application to the server.

If the application has already been deployed locally, *Update* and *Delete* are displayed. If not, *Install* is displayed and *Not Installed* is displayed for the local version number.

1.1.6 Testing Components

You can use the Enterprise Portal Unit Test LaunchPad in the NetWeaver Developer Studio to test your component from within the development environment. The LaunchPad runs your component and displays the output from any test code in the component.

In order to test your component in the LaunchPad, the following is required:

- The component must implement the `com.sapportals.portal.prt.test.ITestable` interface.
- The component must contain one or more test methods. A test method is of the form `testXXX`, where `XXX` is any string, with the following parameters:
 - `IPortalComponentRequest`: The standard portal request object.
 - `IPortalComponentTestResponse`: A special test response object that includes methods for testing conditions and logging the results.

The following is an example of a test method.

```
public void testMyTest(IPortalComponentRequest request,
                      IPortalComponentTestResponse response) {

    INode node = request.getNode();
    NodeMode nodeMode = node.getNodeMode();
    response.assert(nodeMode != INode.EDIT, "EDIT Mode not set!",
    "");
}
```

For more information on using the LaunchPad and creating test code, see [Enterprise Portal Unit Test Studio Perspective \[Page 33\]](#).

1.2 Web Dynpro Applications for the Portal

Purpose

This documentation describes how Web Dynpro application can use SAP Enterprise Portal services and how Web Dynpro applications can be integrated in the SAP Enterprise Portal.

For this documentation you should be familiar with the Web Dynpro development process.

1.2.1 Web Dynpro Java

Purpose

The SAP Enterprise Portal and Web Dynpro for Java are the strategic user interface technologies of SAP and are based on the SAP Web Application Server (WebAS) Java. The SAP Enterprise Portal supports the Web Dynpro application development with functions like:

- Event handling of portal events
- Navigation between Web Dynpro applications within the portal or to any portal content
- WorkProtect mode

Interaction between the Enterprise Portal and Web Dynpro for Java

With Web Dynpro you create interactive Web-based user interfaces for business applications. The portal allows the role-based and secure access to different kinds of information (structured or non-structured), services, and applications using a Web Browser.

Integration of Web Dynpro applications into the Portal

Web Dynpro applications can be integrated into the portal as follows:

- Using a template for the creation of Web Dynpro iViews.
- Accessing your own Web Dynpro administration tools. You can use the Web Dynpro Content Administrator for the Portal content administration.
- Configuring client-side eventing between Web Dynpro applications and between Web Dynpro applications and other portal content.
- Creating single sign-on (SSO) between portal, Web Dynpro, and back-end applications.
- Using the WorkProtect mode of the portal to prevent loss of data when executing Web Dynpro applications.

The portal provides for Web Dynpro applications:

- The navigation in the Web Dynpro-based iViews as well as between them and other portal content (including object-based navigation).
- Web Dynpro applications automatically use the currently set Portal display theme to ensure a consistent appearance

1.2.1.1 Web Dynpro Page Builder

Purpose

The Web Dynpro page builder embeds the Web Dynpro UI technology into the portal platform. Features of the Web Dynpro page builder are:

- Launching a Web Dynpro iView without placing it on a page, for example in the development process of an iView.
- Providing the service factory to the Web Dynpro iView.
- Defining the portal environment needed for the Web Dynpro iView.
- Store the data of explicit and implicit personalization in the Portal Content Directory (PCD).

Web Dynpro Java applications are always rendered in *embedded* mode, without IFrames.

1.2.1.1.1 Creating Web Dynpro Based Portal Content

The portal content administrator creates the portal content from deployed Web Dynpro and other applications. For Web Dynpro applications we have the following iView types:

- Single full-page iView

The Web Dynpro application in a single iView. Single full-page iViews can be personalized and can use services from the service factory but the portal does not know anything about the internal structure of the application.

- Multiple iViews

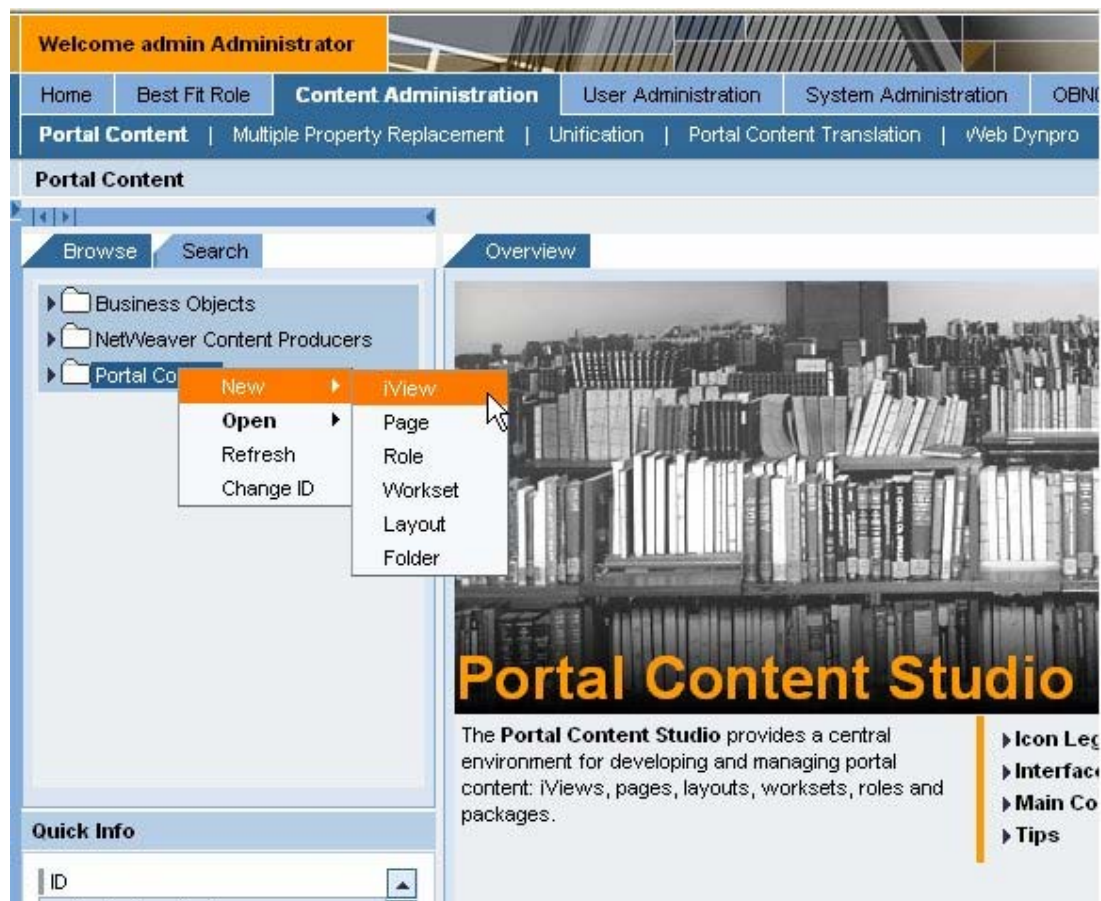
The Web Dynpro application is divided into several iViews. Multiple iViews are necessary when the application has to provide role-based or end-user based customizing of the layout or when features, for example printing, should only be provided for certain parts of the application.

1.2.1.1.1.1 Single Full-Page iView

A single full-page iView is created with the iView wizard.

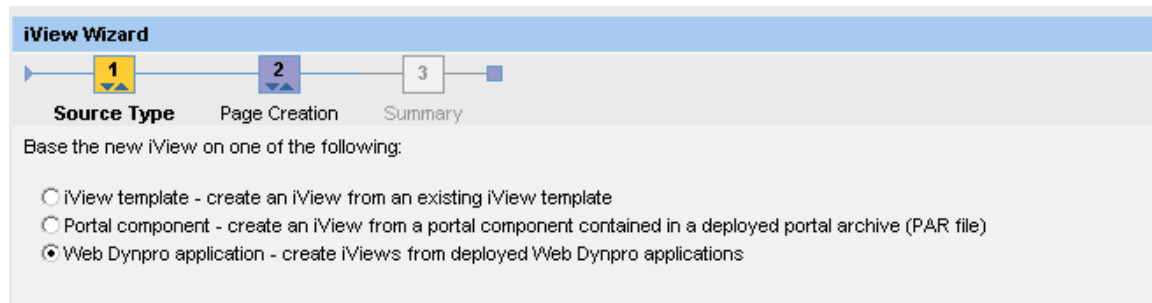
1. Start the iView wizard.

Select the context menu of the folder that should accommodate the created iViews and choose *New* → *iView*:

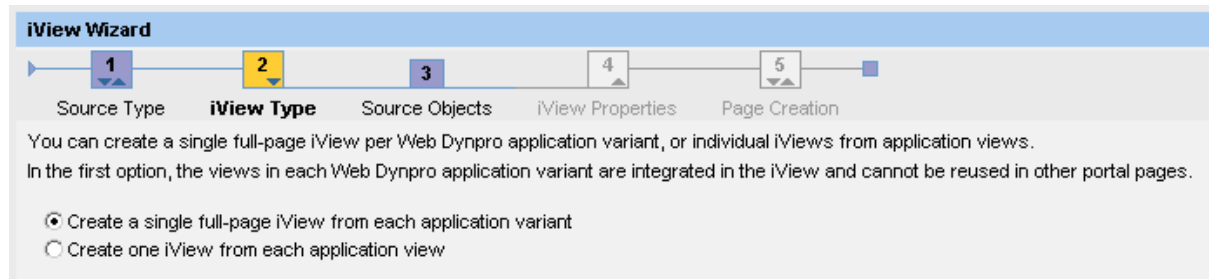


Web Dynpro Applications for the Portal

2. Select the option "Web Dynpro application":

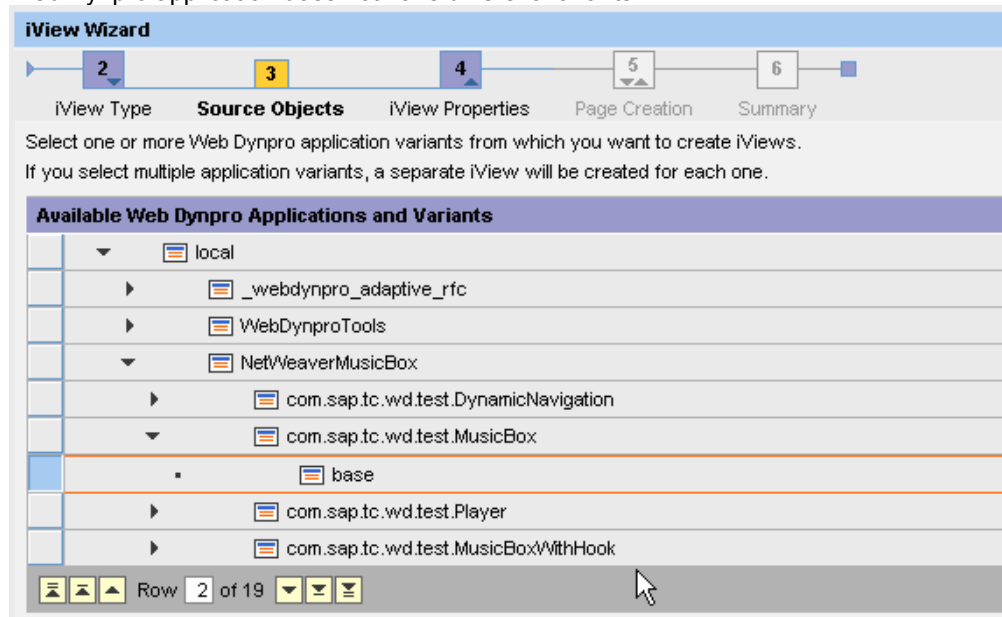


3. Select the option "Create a single full-page iView from each application variant":



4. Choose the variant of a Web Dynpro application that should be placed on the page.

Browse through the deployed Web Dynpro content. The variant "base" indicates that a Web Dynpro application does not have different variants.



5. Modify the iView properties if needed.

The generic iView settings, like the iView name or the technical iView, can be changed.

iView Wizard

2 3 4 5 6

iView Type Source Objects **iView Properties** Page Creation Summary

This screen lists the iViews that will be created. Modify the properties of a selected iView, as needed.

New iView List	
name	base

Row 1 of 1

iView Name: *
MyiView

iView ID: *
java_local_NetWeaverMus

iView ID Prefix (Example: com.companyname):

Master Language *
English

Description:

Apply ID Prefix to All iViews

6. Page Creation.

As an option you can create a Web Dynpro page with the single full-page iView.

iView Wizard

2 3 4 5 6

iView Type Source Objects iView Properties **Page Creation** Summary

You can create a Web Dynpro page and add the new iViews to it.
The new iViews in the page are added as delta links to the new iViews in the Portal Catalog.

☒ Open Page wizard to create a Web Dynpro page

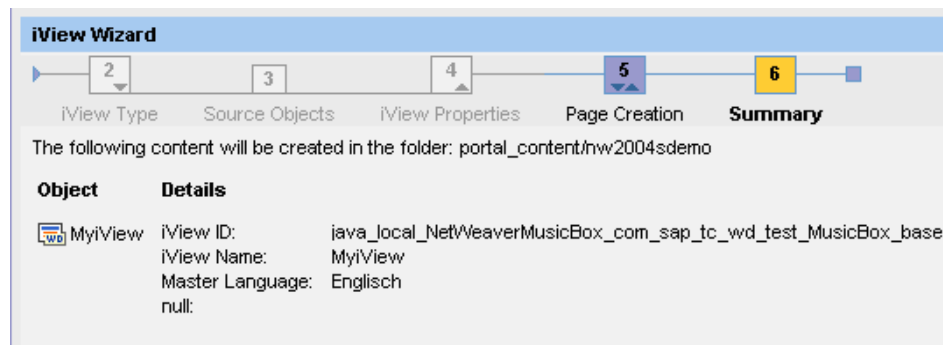


If the option to create a Web Dynpro page around the iView, the unit iViews will be created in the selected folder of the portal catalog. Finally delta links to these unit iViews are created in the portal page.

7. Choose finish to create the iView(s).

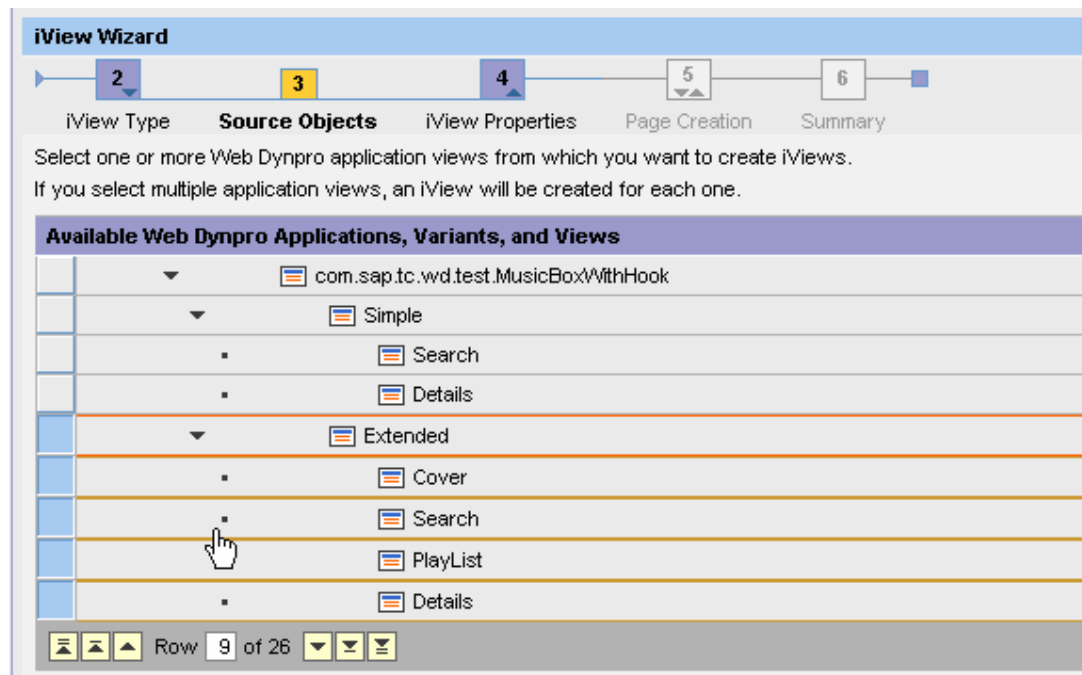
You get a summary of the created portal objects, the iView(s) and the page when the option has been selected (see step 6)..

Web Dynpro Applications for the Portal



1.2.1.1.2 Multiple iViews

Multiple iViews are also created with the iView wizard. Steps 1, 2 and 3 are identical to single full-page iView. In step 4 you can select all the iViews you want to create for the Web Dynpro application. When you select the variant, all iViews provided by this variant are selected.



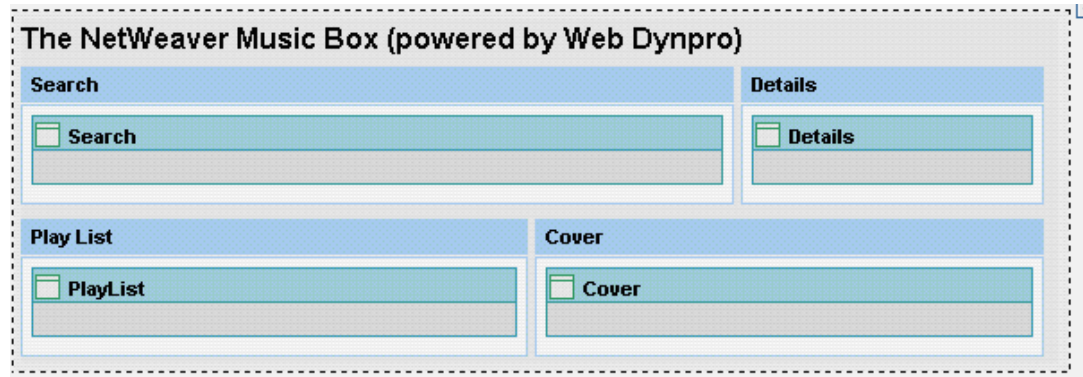
When a Web Dynpro application wants to display more than one iView a static iView list can be used. A static iView list can be declared in the Web Dynpro application as follows:

- Every iView is defined by a `ViewContainerUIElement` object in the root view of the application window.
- The root view can define other UI elements in which the `ViewContainerUIElements` are placed. These additional UI elements are not rendered if the application is split up into several iViews.

Web Dynpro Applications for the Portal

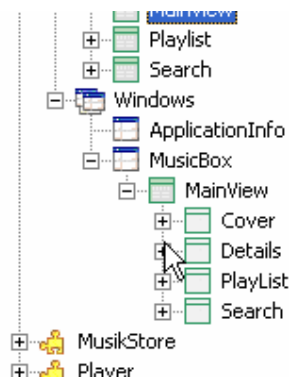
- The used `ViewContainerUIElements` should have describing IDs because the IDs are used as the default `iView` titles.

Example: Root view defines the `iViews` `Search`, `Details`, `PlayList` and `Cover`, placed in a `group` UI element:



The `group` UI element is visible when you start the application as single full-page `iView`, but is not rendered when the application is split up into different `iViews`.

The view hierarchy is displayed as follows ("MainView" as root view):



1.2.1.1.2 Portal Service Factory

With the portal service factory you can access portal services. The portal service factory makes sure that the services are always available, even when the Web Dynpro application and the portal are running on different hosts.

The portal service factory provides the following services:

- Page Service

The page service gets information about the current page layout and content and manipulates the page content by hiding or displaying `iViews`.
- Tray Service

Web Dynpro Applications for the Portal

With the tray service you add Web Dynpro actions to an iView or page tray.

- Navigation Service

With the navigation service you access the object based navigation (OBN) meta data.

1.2.1.1.2.1 Page Service

The page service provides information about the current layout and content of the page running the iView.

Get List of Visible iViews

When a Web Dynpro application has more than one iView, the application can get information which iViews are currently visible on the page.

Example: Get the list of visible iViews.

```
IWDPageService pageService = (IWDPageService)
    WDPortalUtils.getService(WDPortalServiceType.PAGE_SERVICE);

// String array receives iView names
String[] iViews = pageService.getVisibleiViews();
```

Modifying the Page Structure

The page service can hide and display iViews of a Web Dynpro application.

Example:

```
IWDPageService pageService = (IWDPageService)
    WDPortalUtils.getService(WDPortalServiceType.PAGE_SERVICE);

// display iView Search
pageService.showiView("Search");
// hide iView Details
pageService.hideiView("Details");
```

Restrictions

- The layout changes are not saved. When the user navigates to another page and back he will see the original page.
- Only iViews of the current page layout can be displayed or hidden.

Advanced Modifying of the Page Structure

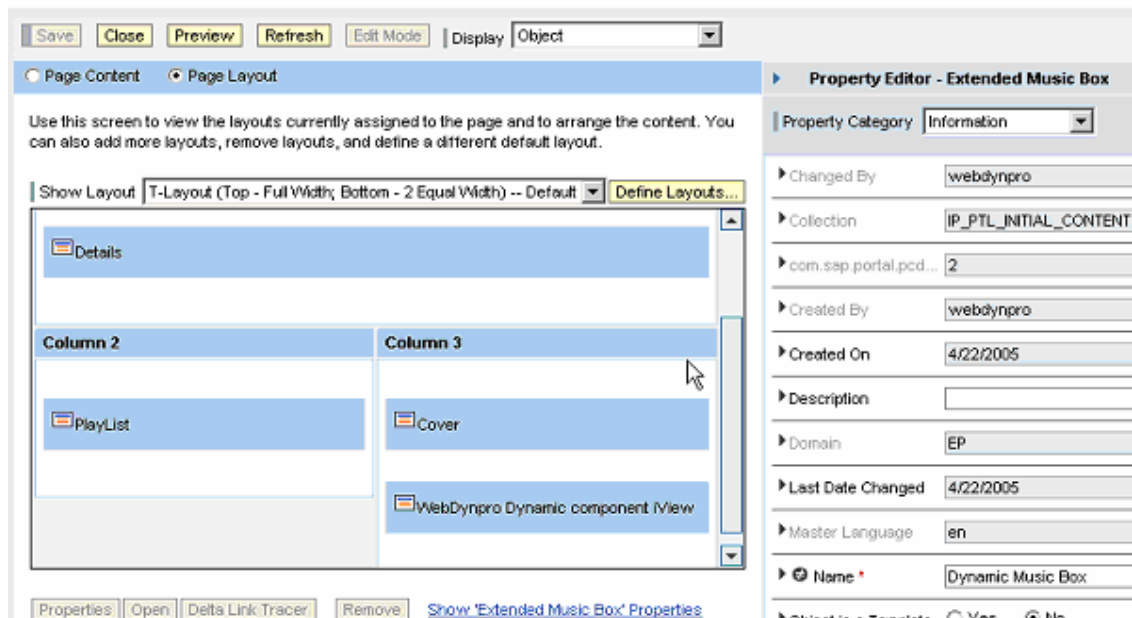
With dynamic iViews you can define place holders in your page layout. The place holder(s) in your page layout are defined with the standard page editor. You have to use the Web Dynpro Dynamic Component iView template that is under:

Portal Content → Content Provided by SAP → Templates → iView Templates

Web Dynpro Applications for the Portal

The iView template is placed like any other iView on your page. With the page layout editor you can define the position of the placeholder iView.

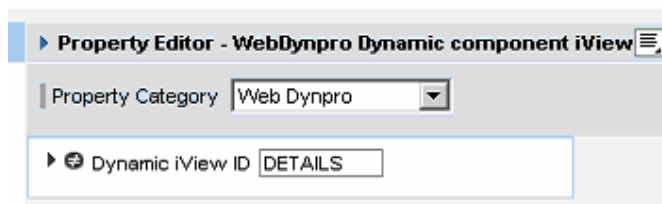
Example:



Every dynamic iView placeholder has a unique ID which is used to identify the placeholder at runtime. The default ID value is *DYNAMIC*. If the page contains only one placeholder you do not have to change the ID.

If you want to change the ID you have to edit the iView properties of the placeholder iView. The "Dynamic iView ID" property is part of the "Web Dynpro" category.

Example: Changing ID to DETAILS:



To replace the defined placeholder or dynamic iView at runtime, you have to specify the iView with the full PCD path.

Example: Replace iView DETAILS.


```

IWDPageService pageService = (IWDPageService)
    WDPortalUtils.getService(WDPortalServiceType.PAGE_SERVICE);

pageService.setDynamicIView(
    "DETAILS",
    "pcd:portal_content/nw2004sdemo/nw2004sdemo/nwmusicbox"+
    "/extendedmusicbox/java_local_NetWeaverMusicBox_com_sap_"+
    "tc_wd_test_MusicBoxWithHook_Extended_Details");

```

Example: Hide iView DETAILS.

```

IWDPageService pageService = (IWDPageService)
    WDPortalUtils.getService(WDPortalServiceType.PAGE_SERVICE);

pageService.hideDynamicIView("DETAILS");

```

The specified iView is added with all its properties to the page embedded or isolated, depending on the iView. The dynamic iView itself does not render any UI.

With the dynamic iView you can extend your Web Dynpro application very convenient to display content that is not created with Web Dynpro, without IFrames.

1.2.1.1.2.2 Tray Service

The tray service allows you to add any Web Dynpro action to an iView tray or to the page tray. These actions are displayed as any other default tray entries of an iView or page tray. After selecting such a tray item the defined Web Dynpro action is triggered for the Web Dynpro application defining the action.

From the application point of view it is absolutely transparent whether the action is triggered by a UI element, which is part of the application UI, or by an iView or page tray. It is also possible to use the same actions for a tray entry and for other UI elements rendered as part of the standard application UI.

Example: Add an action to the iView tray and to the page tray.

```

IWDTrayService trayService = (IWDTrayService)
    WDPortalUtils.getService(WDPortalServiceType.TRAY_OPTION);

if (trayService != null) {
    trayService.addIViewItem("addItemComment", "Add Comment",
        wdThis.wdGetAddItemCommentAction(), null);

    trayService.addPageItem("addItemComment", "Add Comment",
        wdThis.wdGetAddItemCommentAction(), null);
}

```

Usually you call the tray service in the `wdDoModifyView()` method. Please make sure that you call this method only once, otherwise you get duplicate tray entries.

1.2.1.1.3 Personalization

You can define application specific properties, which could be used for role-based customization and/or end-user based personalization.

You can define the scope for every property – some are visible only for the portal content administrator and some are visible for the portal content administrator and for the end-user. The portal content administrator can change the scope for every property, as he can for every other attribute of a page or iView.

1.2.1.1.3.1 Application Specific Property Categories

Application specific property categories help the portal content administrator to handle iViews easier. You can define categories for each single iView provided by one Web Dynpro application or one category for all iViews. If you do not define an application specific property category, the application specific properties are displayed under the generic Web Dynpro category. All PCD property types can be used.

Example: The generic iView editor displays application specific properties for the `NetWeaver Music Box` application.

The screenshot shows a web-based property editor interface. At the top, there are buttons for 'Save', 'Close', 'Preview', 'Refresh', and 'Edit Mode', followed by a 'Display' dropdown menu set to 'Object'. Below this is a section titled 'Property Editor - Search'. Inside this section, there is a 'Property Category' dropdown menu currently set to 'NetWeaver Music Box'. Below the dropdown, there are four property settings: 'Enable Artist Search' with radio buttons for 'Yes' and 'No' (where 'No' is selected), 'Maximum Number of Search Results' with a text input field containing the value '5', 'Root Directory' with a text input field containing the path 'C:\Documents and Settings\ld024318\My Documents\Music', and 'Used Data Source' with a dropdown menu set to 'Default'. A mouse cursor is visible over the 'Used Data Source' dropdown.

In addition to the application specific property categories there is a generic “Web Dynpro” category, containing all technical attributes of a Web Dynpro iView.

Example:

The screenshot shows the 'Property Editor - Search' window. At the top, there are buttons for 'Save', 'Close', 'Preview', 'Refresh', and 'Edit Mode', followed by a 'Display' dropdown menu set to 'Object'. Below this is a 'Property Category' dropdown menu set to 'Web Dynpro'. The main area contains several properties, each with a label and a text input field:

- Application name: MusicBoxWithHook
- Application namespace: local/NetWeaverMusicBox
- Application parameters: (empty field, with a mouse cursor hovering over it)
- Application type: Java
- JNDI context element: Search
- JNDI context element title: Search for Your Music
- Node type: viewarea
- Variant: Extended
- View area: Search

Application parameters are passed to the Web Dynpro application at runtime.

1.2.1.1.3.2 Accessing Application Specific Properties

Once application specific properties are defined the portal content administrator or the end-user can change them. To access the properties from the Web Dynpro application, you have to use the `IViewPersonalization` class. It allows you to access personalization data for every `iView` provided by your Web Dynpro application.

Example:

```

/* In this example we store the returned value in a specific context
 * attribute to make it accessible for each Web Dynpro controller. If there
 * is an error while accessing the personalized data we report an
 * exception. Another option would be of course to use in this case any
 * application defined default values.
 *
 * The IViewPropertyWrapper class provides getter methods for all
 * different types of properties like integer, boolean, translatable text
 * strings or enumerations. */

// Get the iView specific data
IViewPersonalization searchPersData =
    IViewPersonalizationFactory
        .getIViewPersonalization(
            "Search");

// Get the needed property
IViewPropertyWrapper maximumVisibleSearchResults =
    searchPersData.getIViewProperty(
        "NumberOfSearchResults");

if ((maximumVisibleSearchResults != null)
    && (maximumVisibleSearchResults.getIntValue() > 0)) {
    wdCotext
        .currentResultTableElement()
        .setNumberOfVisibleRows(
            maximumVisibleSearchResults.getIntValue());
} else {
    wdComponentAPI.getMessageManager().reportException(
        "Failed to get value of 'NumberOfSearchResults'",
        true);
}

```

1.2.1.1.4 Compatibility

Compatibility to *SAP NetWeaver '04* based Web Dynpro iViews and pages. Compatibility restrictions are:

- *SAP NetWeaver '04* Web Dynpro iViews can be added to a page executed by the Web Dynpro based page builder as isolated iViews only.
- Personalization and all services provided by the portal service factory are not available for *SAP NetWeaver '04* Web Dynpro iViews.
- *SAP NetWeaver 2004s* Web Dynpro iViews can be added to pages executed by the PRT based page builder only as isolated iViews.
- Personalization and all services provided by the portal service factory are not available for *SAP NetWeaver 2004s* Web Dynpro iViews running on a page executed by the PRT based page builder.

1.2.1.2 Portal Events

Purpose

In the *SAP Enterprise Portal*, different application types in specific iViews can be arranged on one page. To communicate between the different iView types the portal provides the Enterprise Portal Client Framework (EPCF), also known as client-side eventing. This document describes how Web Dynpro applications can use EPCF.

Restrictions

Several Web Dynpro applications running on one portal page can communicate using client events. It is recommended, to use client-side eventing only for “occasional” communications between Web Dynpro applications. For Web Dynpro applications that have to interact more frequently, a “full-screen” Web Dynpro application containing all the components has to be implemented.

With SAP NetWeaver '04 Stack 09, client eventing is only supported for the Web Dynpro HTML client.

1.2.1.2.1 Subscribe to a Client Event

The communication between iViews, including Web Dynpro applications, is based on [EPCF \[Page 33\]](#). EPCF uses Javascript to allow iView communication and provides an API that can be used by the portal application developer. Web Dynpro applications have to use a set of Java wrapper methods to implement client-side eventing.

It is possible to subscribe or unsubscribe to certain client events. To do so, you must define which Web Dynpro action is to be used as the event handler for the portal event. You can also fire any portal event.



Because of Javascript restrictions, all participants (the portal server and all used servers) have to be in the same domain when EPCF is used.

The following example demonstrates how to subscribe to a certain portal event.

```
WDPortalEventing.subscribe ("urn:com.sap.tc.webdynpro.test.portal",  
                             "TestEvent",  
                             wdThis.wdGetTestEventAction());
```

You have to define the name space of the event and the name of the event. The combination of these two names must be unique.

The third parameter is the Web Dynpro action that is mapped to the portal event. The event handler is called when the Web Dynpro application receives the specified portal event on the client. The Web Dynpro HTML Client handles the mapping between a portal event and a Web Dynpro action and is absolutely transparent for the Web Dynpro application developer.

You can reuse a Web Dynpro action for several portal events. If you want to receive the transported data of the portal event, you can define the following parameters for your Web Dynpro action:

- dataObject

Web Dynpro Applications for the Portal

This parameter contains the transported parameter of the portal event.

- Namespace

This parameter contains the name space of the received portal event.

- Name

This parameter contains the name of the received portal event.

Adding the nameSpace and name parameters to the Web Dynpro action is useful when the action is reused for several portal events to distinguish the caller.



It is important to remember that in the current version, an event subscription is valid for a Web Dynpro view. Therefore you should add the necessary Java coding, for example, in the `wdDoInit()` method of the generated view class. If you navigate between different views, you have to subscribe every view for the event.

1.2.1.2.2 Unsubscribe a Client Event

Unsubscribing a client event is very similar to subscribing:

```
WDPortalEventing.unsubscribe("urn:com.sap.tc.webdynpro.test.portal",  
                             "TestEvent",  
                             wdThis.wdGetTestEventAction());
```

Make sure that you unsubscribe every single Web Dynpro view, as the subscription and unsubscription is valid only for the current view.

1.2.1.2.3 Raise a Client Event

The following example demonstrates how to raise a portal event:

```
WDPortalEventing.fire ("urn:com.sap.tc.webdynpro.test.portal",  
                      "TestEvent",  
                      "AParameter");
```

You can fire a portal event at any time in your Web Dynpro application. The event is transported with the next response to the client. You can also raise more than one portal event in one request-response cycle. Normally, you will fire a portal event in a Web Dynpro action event handler (for example, pressing a button).

1.2.1.3 Portal Navigation

Purpose

The SAP Enterprise Portal structures the content based on roles. The portal navigation uses the Top Level Navigation (TLN) component and/or the Detailed Navigation Component

Web Dynpro Applications for the Portal

(DTN). Every application running as portal content (ipage or iView) can trigger the portal navigation to iViews or pages.

The integration of the portal navigation features into Web Dynpro is very similar to the [client-side eventing \[Page 33\]](#). Portal application use a Javascript API for the portal navigation. The Web Dynpro runtime offers a `The WDPortalNavigation` service provides methods to use the portal navigation and gives access to the parameters. The `WDPortalNavigation` service is a generic part of the Web Dynpro runtime.

1.2.1.3.1 Absolute Page Navigation

The `WDPortalNavigation` service provides `navigateAbsolute()` methods for absolute page navigation. The methods have the following parameters:

- `navigationTarget`

Specifies the absolute target URL (i.e. the URL to the iView or page acting as navigation destination). This absolute target URL points to the location of the iView or page in the Portal Content Directory (PCD) structure. You have to use the prefix `ROLES://`.

Absolute target URL example:

```
ROLES://portal_content/administrator/super_admin/super_admin_role/com.sap.portal.system_administration/com.sap.portal.support/com.sap.portal.web_dynpro_test_tools/com.sap.portal.portal_navigation.
```

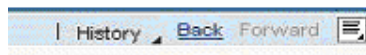
- `mode`

Specifies if the defined navigation target is displayed in the same browser window or in a new one. The following modes are available:

- `WDPortalNavigationMode.SHOW_INPLACE`

The navigation destination is displayed in place, meaning in the same browser window. The Top Level Navigation (TLN) and the Detailed Navigation (DTN) are updated accordingly.

You can use the BACK/FORWARD functionality of the page header to navigate back.



- `WDPortalNavigationMode.SHOW_EXTERNAL`

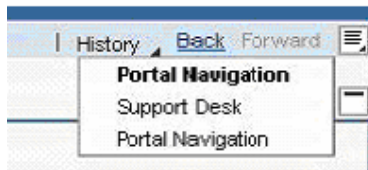
The navigation destination is displayed in a new browser window that has no portal frame. Only the specified iView or page is displayed.

- `WDPortalNavigationMode.SHOW_EXTERNAL_PORTAL`

The navigation destination is displayed in a new browser window with the standard portal frame (containing TLN and DTN). The TLN and DTN are updated accordingly. The new browser window clears the navigation history so you cannot use the BACK/FORWARD functionality of the page header to navigate.

- `historyMode`

The history mode defines how the navigation step is visible in the navigation history as part of the page header.



The following options are available:

- No history

The navigation step is not visible in the navigation history. You cannot use the BACK/FORWARD functionality to navigate back.

- No duplications

The navigation step is visible in the navigation history. If there are several navigation steps to the same navigation destination, they are only visible once in the navigation history.

- Allow duplications

The navigation step is visible in the navigation history. If there are several navigation steps to the same navigation destination, but with different parameters, they are visible as different entries in the navigation history.

This option is useful when you navigate to an iView displaying details of a customer. The customer ID can be a parameter. This results in different entries in the navigation history.

Using the `History Title` parameter as described below, you can define the title of the entry in the navigation history, like "Details for Customer 4711" and "Details for Customer 007" to explain that the same navigation destination is used but with different parameters).

- `targetTitle`

The `targetTitle` defines the title of the entry in the navigation history.

- `contextURL`

Defines the "navigation context" in which the navigation destination should be displayed. The context URL is useful if your specified navigation destination is not visible in TLN or DTN. If you trigger navigation steps to an invisible navigation destination the TLN and DTN is not adjusted. To update the TDN and DTN in this case, you can use the context URL.

The context URL has the same format as the absolute target URL.

- `windowFeatures`

If you start the navigation destination in a new browser window, you can define the used window features, like the window size, the window position, or the visible button bars.

- `windowName`

If you start the navigation destination in a new browser window, you can define the name of this new browser window.

- `launcherParameters`

You can define any parameters for a navigation step. When navigating to SAP-based portal content, you can define parameters that should only be passed to the portal

Web Dynpro Applications for the Portal

component, the `AppIntegrator`, to launch the specific SAP content, like a Web Dynpro application, a BSP application or an IAC-based on the Internet Transaction Server (ITS).

Example how to specify launcher parameters:

```
Namespace=local&ApplicationName=MyTestApp.
```

Depending on the defined parameter values you have to encode the parameter values before assembling the `launcherParameters` string.

- `businessParameters`

If you want to define parameters that are passed to the SAP content itself, you have to define them as business parameters.

Example:

```
CustomerID=4711&DisplayMode=Edit
```

Depending on the defined parameter values you have to encode the parameter values before assembling the `businessParameters` string.

- `useSAPLauncher`

Boolean value. If your navigation destination is content based on SAP technology (i.e. BSP, ITS, BI, or Web Dynpro Java or ABAP) you have to set this flag to "true".

- `postParameters`

Boolean value. If you have several parameters it can be necessary to post the parameters because of URL length restrictions. Setting this flag to "true" will post the parameters.

1.2.1.3.2 Relative Page Navigation

Using an absolute target URL can cause problems if you move the navigation destination.

When you navigate from one page to another page stored in the same content folder, the relative navigation will work even when the entire folder has been moved.

For the relative page navigation the `WDPortalNavigation` service offers the `navigateRelative()` methods. It has following parameters:

- `baseUrl`

This is the starting point of the relative navigation destination. The base target URL has the same format as the absolute target URL of an absolute navigation.

Example of a base URL:

```
ROLES://portal_content/administrator/super_admin/super_admin_role/com.sap.portal.system_administration/com.sap.portal.support/com.sap.portal.web_dynpro_test_tools/com.sap.portal.portal_navigation.
```

- `levelsUp`

You have to define how many levels in the navigation hierarchy you have to go up for the relative navigation.

- `path`

The relative navigation path describes the path to the navigation destination relative to the defined base target URL and the defined number of levels up.

All other (optional) parameters are the same as for the [absolute page navigation \[Page 33\]](#).

1.2.1.3.3 Object Based Navigation (OBN)

Purpose

With the standard [portal navigation \[Page 33\]](#) you define a constant URL. With the OBN you can define an “operation” of a business object.

Example:

You define that you want to trigger the “Display” operation of a “Customer” business object. The specific iView or page that is used to realize or implement this operation is configured within the portal platform and could be role- specific or even user-specific.

For more information about OBN, please refer to the portal OBN documentation.

Like the portal navigation, the Web Dynpro integration of the OBN feature is very similar to the portal eventing integration. The Web Dynpro runtime offers a specific `WDPortalNavigation` service to define the necessary parameters.

1.2.1.3.3.1 Triggering Object Based Navigation

The `WDPortalNavigation` service allows access any page or object-based navigation functionality and parameters from a Web Dynpro application. The `WDPortalNavigation` service is a generic part of the Web Dynpro Runtime.

Triggering Object-Based Navigation

The `WDPortalNavigation` service provides the `navigateToObject()` methods with following parameters:

- **system**
You have to specify the system (alias) to which the business object is assigned. This is a mandatory parameter.
- **businessobjType**
You have to define the business object using this mandatory parameter.

Optional parameters are:

- **objValue**
Normally there are many different instances for one business object.
Example:

A business object called `Customer`. To specify which customer should be used for the object navigation step, you have to specify the object value which is the `customer ID`.

- **operation**
Specifies which operation should be used for the object navigation step.
- **objValueName**
The specified object value is passed as a single URL parameter to the object navigation step. The default name of the parameter is `ObjectValue`. Other parameter names can be specified.
- **businessParameters**
Additional parameters can be specified.

Example:

```
Mode=Edit&ShowHeader=false.
```

Parameters for the object-based navigation destination.

- **forwardOBNMetaData**
With this parameter you can get more information about the current object navigation step.
Example:
An application implements different operations of a business object.
To do that the application has to know which operation was triggered by the object navigation step.
Following parameters are possible:
 - **obn.system**
The system that the business object is assigned.
 - **obn.bo_type**
The business object itself.
 - **obn.operation**
The triggered operation. If the default operation is triggered, the value is `_default_`.

1.2.1.3.3.2 Using the IUserObjectBasedNavigation Service

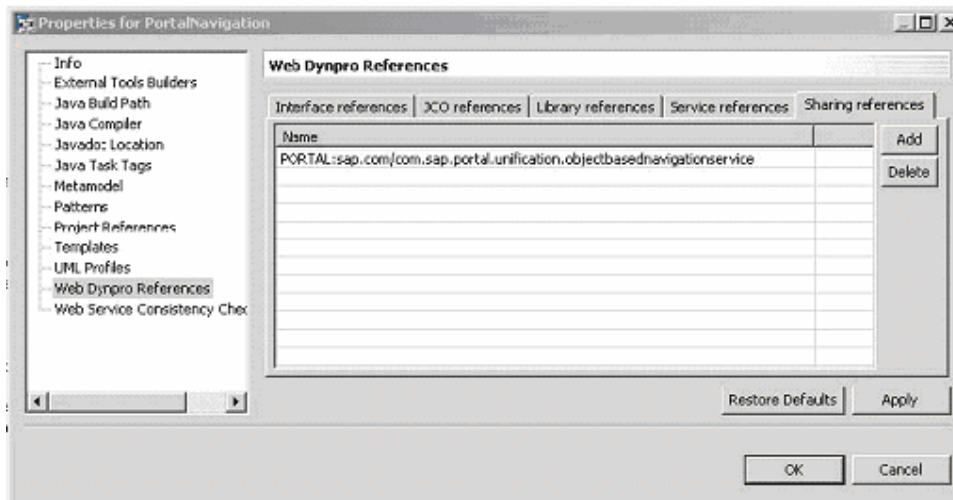
With the [WDPortalNavigation \[Page 33\]](#) service you can trigger an object-based navigation for a operation of a business object. To create a more user friendly user interface you can use the `IUserObjectBasedNavigation` service.

Defining the Necessary Reference

The `IUserObjectBasedNavigation` service is provided by the portal platform. You have to define a specific `sharing reference` in the portal application descriptor file of the Web

Web Dynpro Applications for the Portal

Dynpro application using this service. The following screenshot demonstrates the definition of this sharing reference.



For more information about how to access a portal service see [Accessing an Enterprise Portal Service \[Page 33\]](#).

Operations

This final target URL depends on the configuration for the current role or user. As a result it is possible that for the currently logged in user and his role, there is no target. Such an operation is *invalid*. A *valid* operation is an operation with a defined destination, meaning, there is an iView or page in one of the user roles implementing the requested operation).

Checking a Target

The following code example demonstrates how to check if an operation is valid:

```
// Get a reference to the OBN service
IUserObjectBasedNavigation obnService =(IUserObjectBasedNavigation)
    WDPortalUtils.getServiceReference(IUserObjectBasedNavigation.KEY);

// Get the current user
IUser user = null;
try {
    user = WDClientUser.getCurrentUser().getSAPUser();
} catch (WDUMException e) {
    wdComponentAPI.getMessageManager().reportException(
        "Failed to get current user: " + e.getLocalizedMessage(), true);
}

// Define the system and the business object
String system = "MySystem";
String bo = "customer";

// Call the service
boolean isValidDefaultOperation = obn.isValidTargetExist(system, bo, user);
```

You can use this function to make sure you render the right UI element depending on the configuration for the current user, or to enable or disable a `LinkToAction` UI element that is used to trigger the object-based navigation for the default operation.

Checking a Target for an Operation

The following code example shows how to check if a certain operation has a valid target:

```
// Get a reference to the OBN service
IUserObjectBasedNavigation obnService =(IUserObjectBasedNavigation)
    WDPortalUtils.getServiceReference(IUserObjectBasedNavigation.KEY);

// Get the current user
IUser user = null;
try {
    user = WDClientUser.getCurrentUser().getSAPUser();
} catch (WDUMException e) {
    wdComponentAPI.getMessageManager().reportException(
        "Failed to get current user: " + e.getLocalizedMessage(), true);
}

// Define the system, the business object and the operation
String system = "MySystem";
String bo = "customer";
String operation = "Display";

// Call the service
boolean operationIsValidTarget =
    obn.isValidTargetExistsForOperation(system, bo, operation, user);
```

Getting a List of Valid Operations

The `IUserObjectBasedNavigation` service can be used to get the list of valid operations for a business object.

Web Dynpro Applications for the Portal

Example:

```
// Get a reference to the OBN service
IUserObjectBasedNavigation obnService = (IUserObjectBasedNavigation)
    WDPortalUtils.getServiceReference(IUserObjectBasedNavigation.KEY);

// Get the current user
IUser user = null;
try {
    user = WDClientUser.getCurrentUser().getSAPUser();
} catch (WDUMException e) {
    wdComponentAPI.getMessageManager().reportException(
        "Failed to get current user: " + e.getLocalizedMessage(), true);
}

// Define the system and the business object
String system = "MySystem";
String bo = "customer";

// Get the list of valid operations
List operations = obnService.getTargets(system, bo, user);

// Fill dynamically a context node with the operation information
// The list can be displayed in a DropDownByIndex control.
IMyTestView.IOperationsElement newOperation = null;

for (Iterator iter = operations.iterator(); iter.hasNext();) {
    IOBNTarget target = (IOBNTarget) iter.next();
    newOperation = wdContext.nodeOperations().createOperationsElement();
    newOperation.setCaption(target.getOperationFriendlyName());
    newOperation.setName(target.getOperationName());
    wdContext.nodeOperations().addElement(newOperation);
}
```

Using a Web Dynpro iView as a Target

To make sure that the forwarding of parameters works for the Web Dynpro iView you have to change the Javascript code that is used by the object-based navigation to define the object value manipulation.

For every operation where your Web Dynpro iView is the target (or implementation), you have to define the following Javascript code:

```
return \'DynamicParameter=\' + objValue;
```

Example:

Web Dynpro Applications for the Portal

Object-Based Navigation

Add operations to "pod_portal_content/com.sap.WebDynproTests/com.sap.wdtestrole/com.sap.WebDynproNavigation" by choosing them from the Portal Catalog. This View implements them at runtime.

Operation	Operation Name	Display Name	Business Object	Priority	Relation Resolving
<input checked="" type="radio"/>	Details	Details	Customer	5	None
<input type="radio"/>	Display	Display	Customer	10	None
<input type="radio"/>	Send E-Mail	Send E-Mail	Customer	0	None

Page 1 / 1

Remove Designate Relation Reset to Default

JavaScript for Operation

```
function CBNObjValueManipulation(objValue) {
    return YDynamicParameter=" " + objValue;
}
```

1.2.1.4 Accessing an Enterprise Portal Service

Purpose

Portal services can be used by portal applications, for example, Enterprise Portal Client Framework (EPCF) or the connector framework.

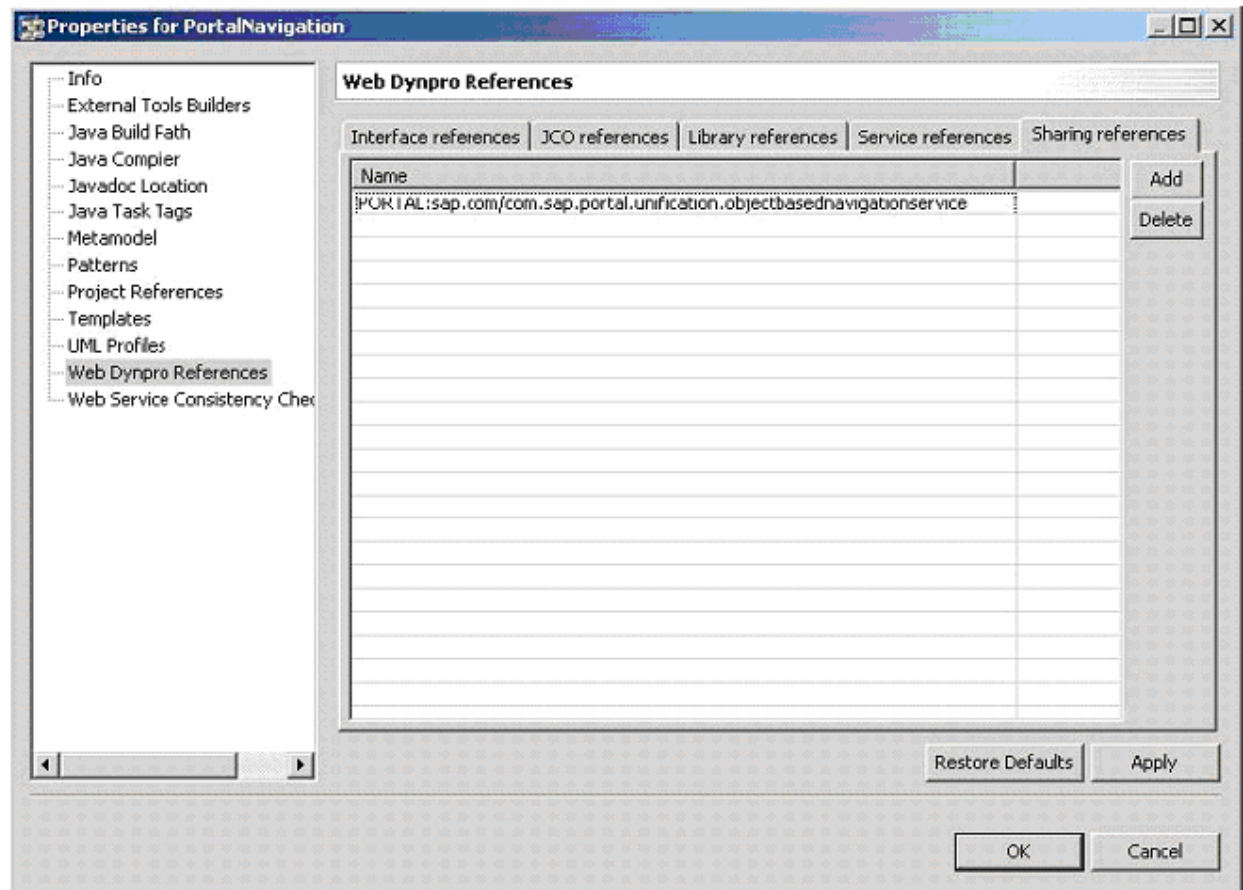
Referencing a Portal Service

Every portal service used in a Web Dynpro application has to be defined as `sharing reference`. To do so, open the "Properties" dialog of your Web Dynpro development component or Web Dynpro Eclipse project when you work with DTR.

The sharing reference must be defined as follows:

```
PORTAL:<Vendor name>/<Full qualified name of the portal service>
```

Example:



After defining the sharing reference the jar file containing the service has to be added to the classpath of the project in the IDE.



Do **not** store the jar file of the service in the `lib` folder of your development component or Eclipse project. This will cause classloader problems at runtime.

Gaining Access to the Portal Service

The following code example demonstrates how to get access the `IUserObjectBasedNavigation` portal service:

```
IUserObjectBasedNavigation obnService = (IUserObjectBasedNavigation)
    WDPortalUtils.getServiceReference(IUserObjectBasedNavigation.KEY);
```

If there is no more than one Web Dynpro controller using the same portal service, you must store this portal service reference in a certain context node, for example, in the component controller context, or any other custom controller context. All controllers using this service must have access using context mapping to the specified context node.

Restrictions

A portal service can be used only when the portal platform is installed and the Web Dynpro application and the portal platform run on the same machine.

Even if you use a complete SAP NetWeaver installation (including the portal platform), you always have to use the local portal service. It is not recommended to use your Web Dynpro-based portal content within a global or federated portal scenario.

1.2.1.5 Using the Work Protect Mode

Purpose

To avoid losing unsaved data when navigating from one iView to another, the SAP Enterprise Portal provides the work-protect mode. Having unsaved data in an application is also called “dirty state”.

Using the Work-Protect Mode

A Web Dynpro application can use the work-protect mode in three levels:

- None
The work-protect mode is not used by the Web Dynpro application.
- Application Only (Default)
Only the Web Dynpro application itself decides if the application has unsaved data. Using the specific work-protect mode Java wrapper class, the Web Dynpro application developer can define the “dirty state” of the application. The “dirty state” is therefore only defined on the server-side. In this level data still can be lost when it was not transported to the server.
- Standard
The Web Dynpro application and the Web Dynpro HTML Client check the application state. Therefore both the application developer and the Web Dynpro HTML Client check the “dirty state” of a Web Dynpro application. The Web Dynpro HTML Client makes sure that no user input data that is typed in but not transported to the server is lost, by setting the “dirty state” of the application in the portal as soon as the user makes an input.

Example: Defining the work-protect mode level.

Web Dynpro Applications for the Portal

```
// Define the needed level. The level can be switched during
// runtime, for example be switched for different views.
WDPortalWorkProtectMode.setApplicationDirtyControl(
    WDApplicationIsDirtyMode.NONE);

WDPortalWorkProtectMode.setApplicationDirtyControl(
    WDApplicationIsDirtyMode.APPLICATION_ONLY);

WDPortalWorkProtectMode.setApplicationDirtyControl(
    WDApplicationIsDirtyMode.STANDARD);
```

Example: Define the “dirty state” of an application:

```
// Set the "dirty state" to YES, i.e. the application is dirty and
// a navigation should be launched in a new window to make sure that
// no data is lost
WDPortalWorkProtectMode.setApplicationDirty(true);

// Set the "dirty state" to FALSE, i.e. the application state is "clean"
// and therefore the navigation is launched in place and the
// running application is left.
WDPortalWorkProtectMode.setApplicationDirty(true);
```

Restrictions

For SAP NetWeaver '04 Stack 09 the following restriction applies:

- The work-protect mode works only for the Web Dynpro HTML Client. All other (rich) clients do not support the work-protect mode.

1.2.1.6 Defining the Theme for Web Dynpro Applications

Purpose

The Web Dynpro application uses automatically the currently selected portal and any user or role-based personalization of the theme.

However, if the Web Dynpro application and the portal are running on different systems with different releases, problems with incompatible style sheets can occur when the portal system has an older release than the system running the Web Dynpro application. This causes errors in the behavior of some UI elements of the Web Dynpro application because the Web Dynpro UI elements also depend on the used themes/style sheets.

Therefore, it is possible to configure the Web Dynpro runtime to use the defined Web Dynpro theme instead of the portal theme. The Web Dynpro application is then rendered in a different theme as the portal, but there is no dependency of the style sheets anymore.

With the Visual Admin tool you can adjust the setting to prevent the Web Dynpro runtime from using the portal theme. Choose `Configuration Adapter` service and navigate to:

webdynpro → sap.com → tc~wd~dispwda → PropertySheet default.

Web Dynpro Applications for the Portal

You have to open this default property sheet and change one property value called:

```
sap.useWebDynproStyleSheet.
```

The possible values are:

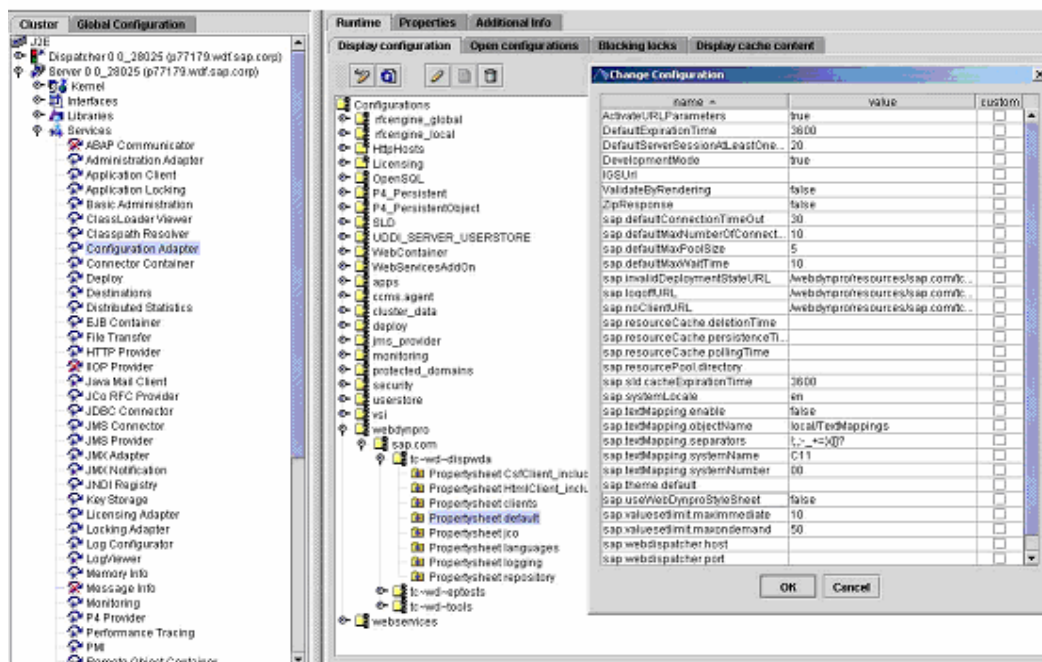
- true

The Web Dynpro application does not use the portal theme but the theme defined by the Web Dynpro runtime.

- false

The Web Dynpro application uses the theme defined by the portal. This is the default for SAP NetWeaver '04 Stack 09.

The following screenshot shows the editor for changing this property value.



Running as a Standalone Application

If the Web Dynpro application runs as a standalone application (not in the SAP Enterprise Portal), or if the Web Dynpro runtime is configured to use the Web Dynpro theme, you have two options to define the theme used by a Web Dynpro application.

Defining the Default Theme

The default theme of the Web Dynpro runtime is changed in the Configuration Adapter service of the Visual Admin tool. Navigate to:

```
webdynpro → sap.com → tc~wd~dispwda → PropertySheet default.
```

Web Dynpro Applications for the Portal

Select the property `sap.theme.default` of the Property sheet. As value, you have to specify a valid, full qualified URL referring the used theme in the form:

```
http://<Your host>:<Your port>/<Your path to the theme>/<Theme
name>
```

Example:

```
http://localhost:50000/irj/portalapps/com.sap.portal.themes.lafs
ervice/themes/portal/sap_chrome.
```

Defining an Application-Specific Theme

If you would like to define a theme only for one specific application, you can use the URL parameter `sap-cssurl` to define the theme. This overrides the default settings. The URL specified has to be valid, full qualified and encoded.

1.2.2 Web Dynpro ABAP

Web Dynpro ABAP applications can be integrated into the SAP Enterprise portal – that is, they can be bound into a portal navigation as an iView. For a description of the individual steps for integrating a Web Dynpro application, see the chapter [Portal Binding: Prerequisites \[Page 33\]](#).

The description given in this programming manual for binding a Web Dynpro application into a portal is written for the developers of applications and is suitable for binding individual applications for test purposes. For more detailed information, refer to the appropriate sections in the *des Portal Development Guide*.

- Using [Portal Events \[Page 33\]](#)
- [Navigation \[Page 33\]](#) between Web Dynpro applications within the portal or to any other portal content.

The following navigation types are supported:

[Object-Based Navigation \[Page 33\]](#)

[Absolute Navigation \[Page 33\]](#)

[Relative Navigation \[Page 33\]](#)

- Using the [Work Protect \[Page 33\]](#) mode

Examples

The following examples of Web Dynpro applications for portal integration are available in the package `SWDP_TEST` in the system:

- `WDR_TEST_PORTAL_EVENT_FIRE`
Trigger event
- `WDR_TEST_PORTAL_EVENT_FIRE2`
Trigger free event

Web Dynpro Applications for the Portal

- WDR_TEST_PORTAL_EVENT_REC
Receive portal event
- WDR_TEST_PORTAL_EVENT_REC2
Receive free portal event
- WDR_TEST_PORTAL_NAV_OBN
Object-based navigation
- WDR_TEST_PORTAL_NAV_PAGE
Page navigation
- WDR_TEST_PORTAL_WORKPROTECT
Security monitoring

1.2.2.1 Binding to Portal: Prerequisites

To be able to integrate a Web Dynpro application, the following prerequisites must be fulfilled.

- You yourself have a user in the portal to which a suitable role is assigned. The role [Content Admin \[External\]](#), for example, contains all the required authorizations and tools. This should always be the case; if not, contact your portal administrator.
- The ABAP system in which the application is located must be known to the portal. This, too, should be the case already. Since a special authorization is required for entering the [system data \[External\]](#), contact your portal administrator to have the ABAP system entered, if necessary. (Documentation about registering the system is available under [Editing SAP System Properties \[External\]](#)). During the following steps, you will need the [system alias \[External\]](#) of the ABAP system that was assigned by the portal administrator in the portal concerned.

For more information on the [different roles \[External\]](#) and the task areas involved, refer to the chapter [Administration Guide for the Portal \[External\]](#).

- To test the application afterwards, you must – of course – also have a user in the ABAP system. Using [user mappings \[Page 33\]](#), you can link your portal user with the ABAP system user in order to avoid a separate logon when calling the application.

As soon as these technical prerequisites are fulfilled, log on to the portal and choose the function *Content Administration* in the [initial navigation screen \[External\]](#). For a description of the following steps, refer to the document [Binding the Application into a Portal \[Page 33\]](#).

1.2.2.2 Integrating an Application in the Portal

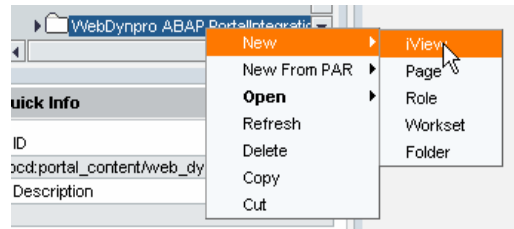
In the navigation panel on the Portal homepage, goto [Content Administration \[External\]](#) and open the folder *portal_content*.

Web Dynpro Applications for the Portal



You can create your application in any folder or insert a separate folder for your application.

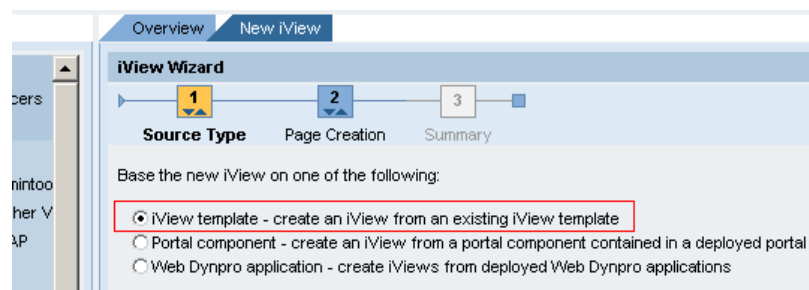
8. Right-click the *portal_content* folder and choose *New* → *iView*.



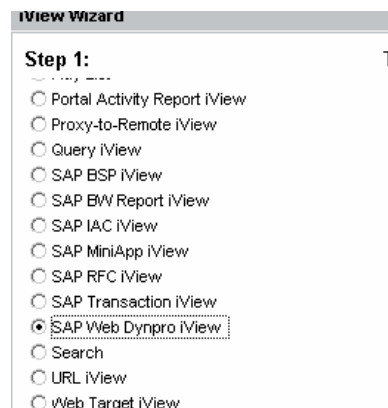
9. The iView wizard opens. Select the source application type and choose *Next*.



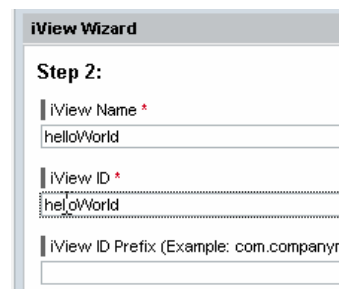
To integrate a Web Dynpro ABAP application, choose **iView template**.



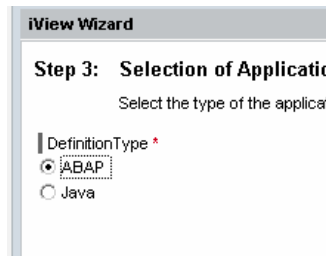
10. From the list choose *SAP Web Dynpro iView* followed by *Next*.



11. Enter a name for your application and choose *Next*.



12. Specify the type, since you want to integrate an ABAP application.



iView Wizard

Step 3: Selection of Application

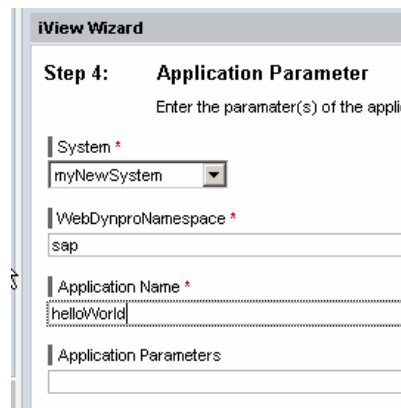
Select the type of the application

DefinitionType *

☒ ABAP

☐ Java

Enter the [System Alias \[External\]](#) and specify the namespace and name of the application. The namespace is **sap**. You can find the application name in the ABAP system.



iView Wizard

Step 4: Application Parameter

Enter the parameter(s) of the application

System *

myNewSystem

WebDynproNamespace *

sap

Application Name *

helloWorld

Application Parameters

13. Choose *Next* and then *Finish*.

Now the application is ready to start.

Additional Settings

You can also make further settings for your application.

Size

Open the iView and choose *Appearance* → *Size* from the dropdown list box.

The standard size of iViews is 80 pixels. This is not large enough for complex applications. Therefore, change the size to *Automatic* or *Full Height*, or increase the number of pixels.

Displaying the iView

The iView has now been created, but you can only display it after setting up the access. Since all applications in the Portal are provided via [Portal Pages \[External\]](#), you will generally have to assign the new iView to a page first. This page is in turn assigned to a role and is called via the role.

However, provided the application is to be called **only for test purposes** by developer himself, the iView can also be assigned to a corresponding test role.



Carry out the following steps to assign the iView to a role.

- a. Open the role by right-clicking and select the folder in the role you want to assign the iView to.

- b. Find the iView in the portal content on the right-hand side.
- c. Right-click to add the iView to the role as a delta link.
- d. Now your iView will appear in the menu structure.

1.2.2.3 Portal Events

In the SAP Enterprise Portal, you can process different application types in special iViews on the same portal page. Here, iViews can be included using different technologies (such as Web Dynpro ABAP, Java, or BSP). The communication between these iViews takes place through an event function – portal eventing (or client-side eventing).

A Web Dynpro ABAP application can be [registered \[Page 33\]](#) for portal events. In this way, the Web Dynpro application can react to an event that was triggered in another iView in the portal. Therefore, it does not matter what technique you used to set up the application that is the basis for the other iView. The assignment as to which event handler is to be called when this event occurs is stored in the Web Dynpro application that has registered itself on the portal event.

Similarly to registration, a Web Dynpro application can [trigger \[Page 33\]](#) any portal event. In this case, the event is passed to the portal by the respective iView. The portal passes the event to all iViews that have registered for this event. The application that finally handles this event can, in turn, have been set up with a different technique than the Web Dynpro application triggering it.



Portal eventing functions only between iViews that are on the same browser window. Events between iViews in different browser windows cannot be transported.

All participating iViews must also belong to the same domain. Otherwise portal eventing cannot work due to JavaScript restrictions.

1.2.2.3.1 Triggering a Portal Event

A portal event is an event that is triggered within an iView – in this case, within a Web Dynpro application – and is then passed by the portal to one or several other iViews. The portal passes the event to all iViews that have registered for this event. In this way, events can be transported between several iViews based on different development techniques.



Portal eventing functions only between iViews that are on the same browser window. Events between iViews in different browser windows cannot be transported.

All participating events must also belong to the same domain. Otherwise portal eventing cannot work due to JavaScript restrictions.

In Web Dynpro ABAP, the Portal-Manager (interface [IF_WD_PORTAL_INTEGRATION \[External\]](#)) provides the FIRE method. Using the [Web Dynpro Code Wizard \[External\]](#), you can insert this method call as a template into your source code and fill it with values in accordance with the requirements of your application.

Web Dynpro Applications for the Portal

```

method ONACTIONFIRE_PORTAL_EVENT .
.
.
.
data: L_API_COMPONENT type ref to IF_WD_COMPONENT,
      L_PORTAL_MANAGER type ref to IF_WD_PORTAL_INTEGRATION.

L_API_COMPONENT = WD_COMP_CONTROLLER->WD_GET_API ( ).
L_PORTAL_MANAGER = L_API_COMPONENT->GET_PORTAL_MANAGER ( ).

L_PORTAL_MANAGER->FIRE(
    PORTAL_EVENT_NAMESPACE = 'my_namespace_for_Web_Dynpro_documentation'
    PORTAL_EVENT_NAME       = 'showCustomer'
    PORTAL_EVENT_PARAMETER = CUSTOM_ID ).

endmethod.

```

In addition to the mandatory parameters *Namespace* and *Name*, you can also pass on another parameter:

Namespace in which the event is stored	'PORTAL_EVENT_NAMESPACE'
Name of the event	'PORTAL_EVENT_NAME'
Parameter	'PORTAL_EVENT_PARAMETER'

You can trigger such a portal event from anywhere in your Web Dynpro application. The event is sent with the next response to the client. You can even trigger several portal events in one request-response cycle.

However, it is usual to trigger a portal event in an action handler of a Web Dynpro application. This could be the case, for example, with an action handler of a UI element (for example, a button). When a portal event is triggered, an internal application event is first passed from the iView to the portal and can be handled within one or several other iViews.

Syntax for Namespace and Names of Events

The characters permitted for the namespace and event name are restricted to the namespaces of the SAP Enterprise Portal – Client Framework.

You can only use the characters listed in the table below.

Valid Characters

Uppercase letters	"A" - "Z"
Lowercase letters	"a" - "z"
Numbers	"0" - "9"
Special character	"- " _ " ."

Also note that:

- The namespace must begin with the character string **urn:**
- Namespaces **com.sapportals.portal.*** and **com.sapportals.*** are reserved for SAP, and therefore you should not use them for your applications.
- Note that the namespace and the name are case-sensitive.



urn:com.sap.webdynpro.testApplications.testEvent

Example

- You can find examples in the system under the Web Dynpro applications:
WDR_TEST_PORTAL_EVENT_FIRE
Trigger Event
- WDR_TEST_PORTAL_EVENT_FIRE2
This application serves as a test application. You can enter the name of an event from your own application in order to test the event separately.

1.2.2.3.2 Registering and Handling an Event

Registration for a Portal Event

To register your Web Dynpro application for a portal event, you have the method SUBSCRIBE_EVENT available in the interface [IF_WD_PORTAL_INTEGRATION \[External\]](#).



To delete your registration for the portal event, use the method UNSUBSCRIBE_EVENT in the portal manager.



Registration or deletion of registration must be performed individually for each view in the respective method WDDOINIT.

Generate yourself a suitable template using the [Web Dynpro Code Wizard \[External\]](#). You can then fill this with values.

```
method WDDOINIT .
    data: L_API_COMPONENT type ref to IF_WD_COMPONENT,
          L_PORTAL_MANAGER type ref to IF_WD_PORTAL_INTEGRATION,
          VIEW type ref to IF_WD_VIEW_CONTROLLER.

    L_API_COMPONENT = WD_COMP_CONTROLLER->WD_GET_API ( ).
    L_PORTAL_MANAGER = L_API_COMPONENT->GET_PORTAL_MANAGER ( ).

    VIEW ?= WD_THIS->WD_GET_API ( ).

    L_PORTAL_MANAGER->SUBSCRIBE_EVENT(
        PORTAL_EVENT_NAMESPACE =
        'my_namespace_for_Web_Dynpro_documentation'
        PORTAL_EVENT_NAME      = 'showCustomer'
        VIEW                   = VIEW
        ACTION                  = 'RECEIVE_CUSTOMER_ID' ).

endmethod.
```

Enter the namespace and the name of the event. The combination of namespace and event name must be unique. In addition, enter the name of the action that is to be triggered if exactly this portal event is to be received. The corresponding action handler is then called automatically.



The action, in this case RECEIVE_CUSTOMER_ID, is not created automatically. Therefore, create the action explicitly on the tab page *Actions* in the view.

Handling a Portal Event

The parameters of a portal event are passed to the action parameter WDEVENT using its method GET_STRING. With the help of the optional parameter PORTAL_EVENT_PARAMETER, you can have application-dependent information passed to the handler method. In the following example, this is the ID of a particular customer whose value is passed to the SHOWCUSTOMER method of the component controller called afterwards.

```
method ONACTIONRECEIVE_CUSTOMER_ID .
    data: EVT_NAME type STRING,
          CUST_ID type SCUSTOM-ID.

    EVT_NAME = WDEVENT->GET_STRING( NAME = 'PORTAL_EVENT_NAME' ).

    if EVT_NAME = 'showCustomer' .
        CUST_ID = WDEVENT->GET_STRING( NAME = 'PORTAL_EVENT_PARAMETER' ).
        WD_COMP_CONTROLLER->SHOWCUSTOMER( CUSTOMER_ID = CUST_ID ).
    endif.
endmethod.
```

Example

You can find examples in the Web Dynpro applications in the system:

- WDR_TEST_PORTAL_EVENT_REC
Receive portal event
- WDR_TEST_PORTAL_EVENT_REC2

This application also serves as a test application. You can enter the name of an event from your own application in order to test the event separately.

1.2.2.4 Portal Navigation

The SAP Enterprise Portal supports navigation between various types of portal content. For example, a Web Dynpro application can navigate to the portal content as well as to another Web Dynpro application that is set up differently. Portal content can be, for example, a BSP or an ITS application.

As well as being [object-based \[Page 33\]](#), page navigation can be [absolute \[Page 33\]](#) or [relative \[Page 33\]](#).

1.2.2.4.1 Object-Based Navigation (OBN)

The structure of SAP Enterprise Portal content is based on roles. You can browse through the user-specific navigation structures using the top-level navigation and the detailed navigation. Portal navigation allows you to navigate between different iViews or pages in any application running as a portal content (that is, any page or iView).

In many cases it is sufficient to use either relative or absolute navigation to navigate to a specific iView or page. However, sometimes more flexibility is required. For this purpose, you have [Object-Based Navigation \[Page 33\]](#), which allows you to define navigation steps at a higher semantic level. Instead of defining a concrete target URL, you call a particular operation of a particular [business-object \[Page 33\]](#).

You configure in the portal which concrete iView (or which page) is to be used for executing this operation. This configuration can be role-specific or user-specific. The Web Dynpro application itself passes on solely the name of the business object and the operations linked to it.

In Web Dynpro for ABAP, the integration of object-based navigation is very similar to the integration of [portal eventing \[Page 33\]](#). To trigger the navigation itself, the Web Dynpro Framework provides a service that can be called from the application. This service, like Portal Eventing, is part of the portal manager.

Triggering Object-Based Navigation

You can activate object-based navigation of the portal in Web Dynpro ABAP by calling the method `NAVIGATE_TO_OBJECT` of the portal manager (interface [IF_WD_PORTAL_INTEGRATION \[External\]](#)). You can generate an appropriate template using the [Web Dynpro Code Wizard \[External\]](#), in which you then enter values.

```
data LR_COMPONENTCONTROLLER type ref to IG_COMPONENTCONTROLLER .
data L_API_COMPONENTCONTROLLER type ref to IF_WD_COMPONENT.
data LR_PORT_MANAGER type ref to IF_WD_PORTAL_INTEGRATION.

LR_COMPONENTCONTROLLER = WD_THIS->GET_COMPONENTCONTROLLER_CTR( ).
L_API_COMPONENTCONTROLLER = LR_COMPONENTCONTROLLER->WD_GET_API( ).
LR_PORT_MANAGER = L_API_COMPONENTCONTROLLER->GET_PORTAL_MANAGER( ).

call method LR_PORT_MANAGER->NAVIGATE_TO_OBJECT
exporting
  SYSTEM                = NAVIGATION_DATA-SYSTEM
  OBJECT_TYPE            = NAVIGATION_DATA-OBJECT
  OPERATION              = NAVIGATION_DATA-OPERATION
  OBJECT_VALUE_NAME      = NAVIGATION_DATA-OBJECT_VALUE_NAME
  OBJECT_VALUE           = NAVIGATION_DATA-OBJECT_VALUE
  BUSINESS_PARAMETERS    = BUS_PARAMETER_LIST
  FORWARD_OBN_METADATA  = NAVIGATION_DATA-FORWARD_OBN_METADATA.
```

Only two parameters are required for the navigation:

- **SYSTEM**
Specify the system (or the system alias) the business object is assigned to.

Web Dynpro Applications for the Portal

- OBJECT_TYPE

Specify the business object you are using.

All other parameters are optional.

- OBJECT_VALUE

Usually there are many different instances of a business object – for example, for the business object *Customer*. You use this parameter to specify which specific customer (for instance, use the customer number) you want to use for the object-based navigation step.

- OPERATION

You use this parameter to specify which operation is to be used for the object-based navigation step.

- OBJECT_VALUE_NAME

The specified object value is transferred as a URL parameter to the OBN step. The standard name of this parameter is *ObjectValue*. You can change this name if you want.

- BUSINESS_PARAMETERS

As well as specifying the object value you can define other parameters that are to be forwarded by the OBN step. An example of a parameter string you could define is:

Mode=Edit&ShowHeader=false.

These parameters can be used by the target of the OBN if the operation of the business object has been prepared accordingly (see below under the section *Maintaining the Target Application in the Portal*).

- FORWARD_OBN_METADATA

Sometimes it is useful for the OBN target to receive more details about the current navigation step. For instance, if you implement an application that serves for implementing different operations performed on a business object, the application must know which operation was triggered by the OBN step. Therefore, you can pass on the following parameters:

- obn.system

The system the business object is assigned to.

- obn.bo_type

The business object itself

- obn.operation

The respective operation If the default operation is used, the value is `_default_`.

Maintaining the Target Application in the Portal

The target application is maintained in the portal for the respective operation of the business object. This is usually done in by the portal administrator.

To be able to transport the business parameters correctly from a Web Dynpro ABAP application to the target application, the following JavaScript must be stored at the target application under *Object-Based Navigation*:

```
return 'DynamicParameter=' + objValue;
```

Example

You can find an example in the system in the Web Dynpro application, WDR_TEST_PORTAL_NAV_OBN.

Role-Based Authorization Check

Execution of the navigation is thus dependent on the customizing for the role settings in the portal. For example, the user of a role could have the authorization for displaying and editing the content of a page, while the users of another role might only be allowed to display the content. If a user triggers object-based navigation, but does not belong to a role that has authorization for the respective operation, an appropriate error message will appear. So as to make the user interface as user-friendly as possible, it is a good idea – from the very start – not to provide this operation for the user in question. For this purpose, however, the information for the authorization of the respective operation must be got by the portal. This can be done with the help of a Web service that is called from the Web Dynpro ABAP application using the class [CL_WDR_PORTAL_OBNWEB_SERVICE \[External\]](#).

1.2.2.4.2 Absolute Navigation

You can activate absolute path navigation for the portal in Web Dynpro ABAP using the portal manager (interface [IF_WD_PORTAL_INTEGRATION \[External\]](#), method `NAVIGATE_ABSOLUTE`). You can generate a template using the wizard, in which you then enter values.

With the absolute navigation tool, you must know the name of the page to be displayed in order to pass it to the method.

```
data LR_COMPONENTCONTROLLER type ref to IG_COMPONENTCONTROLLER .
data L_API_COMPONENTCONTROLLER type ref to IF_WD_COMPONENT.
data LR_PORT_MANAGER type ref to IF_WD_PORTAL_INTEGRATION.

LR_COMPONENTCONTROLLER = WD_THI S->GET_COMPONENTCONTROLLER_CTR( ).
L_API_COMPONENTCONTROLLER = LR_COMPONENTCONTROLLER->WD_GET_API( ).
LR_PORT_MANAGER = L_API_COMPONENTCONTROLLER->GET_PORTAL_MANAGER( ).

call method LR_PORT_MANAGER->NAVIGATE_ABSOLUTE
  exporting
    NAVIGATION_TARGET      = NAVIGATION_DATA-TARGET
    NAVIGATION_MODE        = NAVIGATION_DATA-NAVIGATION_MODE
    WINDOW_FEATURES        = NAVIGATION_DATA-WINDOW_FEATURES
    WINDOW_NAME            = NAVIGATION_DATA-WINDOW_NAME
    HISTORY_MODE           = NAVIGATION_DATA-HISTORY_MODE
    TARGET_TITLE           = NAVIGATION_DATA-TARGET_TITLE
    CONTEXT_URL            = NAVIGATION_DATA-CONTEXT_URL
    POST_PARAMETERS        = ABAP_FALSE
```

Web Dynpro Applications for the Portal


USE_SAP_LAUNCHER	=	ABAP_TRUE
BUSINESS_PARAMETERS	=	BUS_PARAMETER_LIST
LAUNCHER_PARAMETERS	=	LAUNCHER_PARAMETER_LIST.

The navigation target is the only parameter required here. It stands for an absolute address in the portal. The other parameters are used for controlling the navigation and are optional. You can set business parameters and parameters for the respective application launcher in the portal. To transport business parameters correctly to the target application, you have to set the parameter USE_SAP_LAUNCHER. If it is an SAP application (for example, BSP Web Dynpro, and so on), you have to set the switch to TRUE.

Overview of Parameters

Name	Optional	Possible values		Description
NAVIGATION_TARGET		Address, for example ROLES://portal_content/ web_dynpro_abap/ web_dynpro_abap_tester/ portal_integration/ portalNavigation/ portal_navigation_target		Absolute address, path f in the portal content dire This path is displayed in catalog – for instance, w page or an iView.
NAVIGATION_MODE	√	"INPLACE"	Displays the navigation target on the same page	Navigation mode
		"EXTERNAL"	Displays the navigation target on a new page, but only as an iView, without the portal	
		"EXTERNAL_PORTAL"	Displays the navigation target on a new portal page.	
WINDOW_FEATURES	√	"TOOLBAR"	Displays the standard toolbar	Additional JavaScript pa external window – for ex character set or size spe as width=300 or he These parameters are s commas. Spaces are no
		"LOCATION"	Displays the Web address	
		"DIRECTORIES"	Displays the directory buttons of the browser	
		"STATUS"	Displays the status bar	
		"MENUBAR"	Displays the menu bar of the browser	
		"SCROLLBARS"	Displays the scroll bar	
		"RESIZABLE"	Windows can be resized	
		"WIDTH"	Width of the window	
		"HEIGHT"	Height of the window	
WINDOW_NAME	√	String		Title of the target page ir browser. The specified V loaded into a window of instance, MyWindowNar

Web Dynpro Applications for the Portal

				used to access the address
HISTORY_MODE	√	"ALLOW_DUPLICATIONS"	A navigation entry can be listed more than once in the history.	Specifies whether the visited address should be listed in navigation history.
		"NO_DUPLICATIONS"	A navigation entry can be listed only once in the history.	
		"NO_HISTORY"	No navigation entry in the history	
TARGET_TITLE	√	String		Title of the portal page
CONTEXT_URL	√	String		Determines the navigation target
POST_PARAMETERS	√	"TRUE"	Transfer parameters as a POST request	Transfer options for parameters
		"FALSE" (default value)	Transfer parameters as a GET request	
USE_SAP_LAUNCHER	√	"TRUE" (default value)	Target is called using the SAP launcher – for example, BSP	SAP launcher is used
		"FALSE"	Target is not called using the SAP launcher	
BUSINESS_PARAMETERS	√	See structure WDR_NAME_VALUE_LIST with name and value pairs		Transfer parameters for target application (Web Dynpro Web application), for example, customer number. These are transferred by URL. See also URL Parameters  Keep in mind the transfer size, for example, a parameter larger than 1 KB.
LAUNCHER_PARAMETERS	√	See structure WDR_NAME_VALUE_LIST with name and value pairs		Parameter list for the application launcher, parameter list WebDynproNamespaces



If you define BUSINESS_PARAMETERS as application parameters in your Web Dynpro application and the parameter names start with "APP", they will automatically be forwarded to the startup plugs of the Web Dynpro application – provided they are marked as startup parameters. In this case, keep in mind that the iView/page used as the navigation target must be assigned to the user role. If it is not, navigation cannot be triggered.

Example

You can find an example in the system in the Web Dynpro application, WDR_TEST_PORTAL_NAV_PAGE.

1.2.2.4.3 Relative Navigation

The relative path navigation variant can be used, for example, for delivering content across several directories.

To activate relative path navigation for the portal in Web Dynpro ABAP, use the portal manager ([IF WD PORTAL INTEGRATION \[External\]](#), method NAVIGATE_RELATIVE).

You can generate a template using the [wizard \[External\]](#), in which you then enter values.


```
call method LR_PORT_MANAGER->NAVIGATE_RELATIVE
exporting
  BASE_URL           = NAVIGATION_DATA-BASE_URL
  LEVELS_UP          = LEVELS_UP
  PATH               = PATHLIST
  NAVIGATION_MODE    = NAVIGATION_DATA-NAVIGATION_MODE
  WINDOW_FEATURES    = NAVIGATION_DATA-WINDOW_FEATURES
  WINDOW_NAME        = NAVIGATION_DATA-WINDOW_NAME
  HISTORY_MODE       = NAVIGATION_DATA-HISTORY_MODE
  TARGET_TITLE       = NAVIGATION_DATA-TARGET_TITLE
  CONTEXT_URL        = NAVIGATION_DATA-CONTEXT_URL
  USE_SAP_LAUNCHER   = ABAP_TRUE
  BUSINESS_PARAMETERS = BUS_PARAMETER_LIST
  LAUNCHER_PARAMETERS = LAUNCHER_PARAMETER_LIST.
```

Relative navigation can be used from the defined starting point.

Overview of Parameters

Name	Optional	Possible values	Description
BASE_URL	√	Path name	Starting point for relative navigation
LEVELS_UP		Numeric value	Number of navigation steps upwards in the navigation structure
PATH		Path name	Relative path list for the target application

You use the remaining parameters in the same way as for [Absolute Navigation \[Page 33\]](#).

 **BASE_URL :**
pcd:role1/folder1/folder2/fodler3/workset1/page1

LEVELS_UP: 3

PATHLIST: folder4/workset2/page2

The target is:
pcd:role1/folder1/folder2/folder4/workset2/page2

The **BASE_URL** does not have to be specified. If it is not, the current URL is used. **PATH** is a list of the nearest nodes. In this case, these would be:

folder4

workset2


page2

Example

You can find an example in the system in the Web Dynpro application **WDR_TEST_PORTAL_NAV_PAGE**.

1.2.2.5 Work Protect Mode

The work protect mode provides the infrastructure for handling unsaved data in SAP Enterprise Portal. An application is called “dirty” if the entered data has not yet been saved. Normally data is lost when the user navigates to another application without having first saved the data. To prevent this from happening, the client framework of the portal monitors the current status of all the applications in the portal.

 Example of dialog box showing the work protect mode:



The application must define a special status (dirty flag), which tells the portal when there is unsaved data.

You can set and cancel this status (TRUE, FALSE) using method **SET_APPLICATION_DIRTY_FLAG** in interface **IF_WD_PORTAL_INTEGRATION**. If the dirty flag is set to TRUE, each navigation step is automatically executed in a new window. The unsaved data is retained in the original window. This means the user can switch to the original application and save the data afterwards.

The following source text shows how to set the dirty status:

```
data L_COMPONENTCONTROLLER type ref to IG_COMPONENTCONTROLLER .
data L_API_COMPONENTCONTROLLER type ref to IF_WD_COMPONENT.
```

Example

```

data L_PORTAL_MANAGER type ref to IF_WD_PORTAL_INTEGRATION.
  L_COMPONENTCONTROLLER = WD_THIS->GET_COMPONENTCONTROLLER_CTR( ).
  L_API_COMPONENTCONTROLLER = L_COMPONENTCONTROLLER->WD_GET_API( ).
  L_PORTAL_MANAGER = L_API_COMPONENTCONTROLLER->GET_PORTAL_MANAGER( ).

call method L_PORTAL_MANAGER->SET_APPLICATION_DIRTY_FLAG
  exporting
    DIRTY_FLAG = TRUE | FALSE
.
```

Web Dynpro supports the work protect mode in three different ways (method SET_WORK_PROTECT_MODE in the interface IF_WD_PORTAL_INTEGRATION):

- NONE

This value means that the work protect mode is not used by the Web Dynpro application. If you navigate to another application in the portal, unsaved data is lost, even if you set the dirty flag.

- APPLICATION_ONLY

This value means that the Web Dynpro application itself decides if there is still unsaved data – that is, whether the application is dirty. This is why the “dirty” status is only monitored by the server. With this value you cannot ensure that data that has not yet been transferred to the server will not be lost.

- BOTH

This value means the client also checks the “dirty” status. This ensures that no user input that has not yet been transferred to the server will be lost. This is done by setting the dirty status of the application in SAP Enterprise Portal as soon as the user has entered data.

The modes described above can be changed as often as required during runtime of a Web Dynpro application. For example, you can change the mode when a user navigates from one view to another. For one view, it may make sense to save data that is entered in an input field. In this case you define the value BOTH or APPLICATION_ONLY. For another view this protection mode may not be necessary. In this case you define the value NONE.

The source text below shows how the work protect mode can be set:

```

data L_COMPONENTCONTROLLER type ref to IG_COMPONENTCONTROLLER .
data L_API_COMPONENTCONTROLLER type ref to IF_WD_COMPONENT.
data L_PORTAL_MANAGER type ref to IF_WD_PORTAL_INTEGRATION.
  L_COMPONENTCONTROLLER = WD_THIS->GET_COMPONENTCONTROLLER_CTR( ).
  L_API_COMPONENTCONTROLLER = L_COMPONENTCONTROLLER->WD_GET_API( ).
  L_PORTAL_MANAGER = L_API_COMPONENTCONTROLLER->GET_PORTAL_MANAGER( ).

call method L_PORTAL_MANAGER->SET_WORK_PROTECT_MODE
  exporting
    MODE = NONE | APPLICATION_ONLY | BOTH
.
```

1.3 Example

You can find an example in the system in the Web Dynpro application WDR_TEST_PORTAL_WORKPROTECT.

1.4 SAP NetWeaver Developer Studio Plug-In

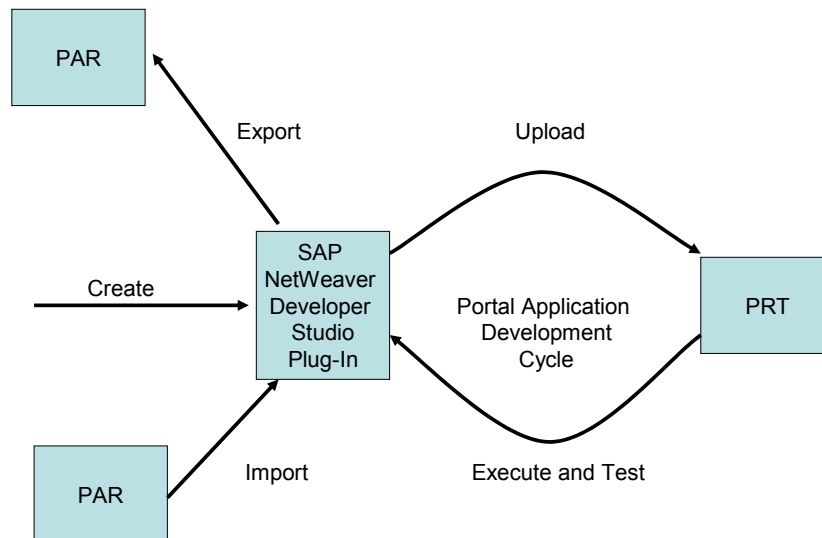
Purpose

The SAP NetWeaver Developer Studio's Enterprise Portals plug-ins enables you to:

- Add a portal object to a PAR file. Supported objects are portal components, portal services, portal Web services.
- Create a portal Web service from a WSDL file or portal service.
- Export, import and create PAR files.
- Edit the code, compile and add resources to portal applications.
- Upload, execute or debug portal applications in one environment.

Implementation Considerations

Portal Application Development Model



A portal application has to be packaged in a PAR file and must be uploaded in the portal before it is executed.

See also:

- [Configuring the Plug-In \[Page 33\]](#)
- [Managing Enterprise Portals \[Page 33\]](#)
- [Managing PAR and JAR Files in the Project \[Page 33\]](#)
- [Enterprise Portal Unit Test Studio Perspective \[Page 33\]](#)
- [Enterprise Portal Web Services Checker Description \[Page 33\]](#)

- [Development Configuration \[Page 33\]](#)
- [Registering an Additional Plug-in \[Page 33\]](#)

1.4.1 Configuring the Plug-In

Use

The SAP Enterprise Portal requires information to be able to upload a portal application to the portal.

Procedure

Adding or Removing the Plug-in Toolbar to the Developer Studio

1. From the menu path, choose *Window → Customize Perspective...*
2. Select *Other → SAP Portal Actions*.
3. To add the toolbar, select the checkbox.
4. To remove the toolbar, deselect the checkbox.

Configuring Enterprise Portals

1. From the menu path choose *Window → Preferences → SAP Enterprise Portal:*
2. Choose *Add* and specify the properties of the portal.
3. If you want your password to be remembered upon deployment; choose *Yes, I want to remember passwords for deployment*.
4. If you want to set this portal installation as the default for deployment, select the checkbox in the *Default* field of the configuration list.
5. Apply the settings to the SAP NetWeaver Developer Studio.

Configuring the Project Settings


From the menu path choose *Window → Preferences → SAP Enterprise Portal → Application Development Studio*.

If You Want To	Then
Automatically reference basic libraries, required for portal development	Choose <i>Yes, I want the default libraries for each new plugin generated project</i>
Specify that the non-Java source files should be included in the generated JAR files	Choose <i>Yes, I want the non Java resources included in the portal components jars</i>
Specify that the generated JAR files should have shorter names, instead of <code><projectname>api.jar</code>	Choose <i>Yes, I want short jar names in the par archive</i>

1.4.2 Managing Enterprise Portals

Propose	Procedure
Create a new portal application project	<ol style="list-style-type: none"> 1. From the menu path choose <i>File</i> → <i>New</i> → <i>Project...</i> → <i>Portal Applications</i> → <i>Create a Portal Application Project</i> 2. Specify the project name and folder that will contain the project. 3. Choose <i>Finish</i>.
Create a new portal service	<ol style="list-style-type: none"> 1. From the menu path, choose <i>File</i> → <i>New</i> → <i>Other...</i> 2. In the dialog that appears select <i>Portal Application</i> → <i>Create a new Portal Application Object</i>. 3. Choose <i>Next</i>. 4. Select the project to create the service in. Choose <i>Next</i>. 5. Select <i>Portal Service</i> and choose <i>Next</i>. 6. Enter data as required. 7. Choose <i>Finish</i>.
Create new portal component	<ol style="list-style-type: none"> 1. From the menu path, choose <i>File</i> → <i>New</i> → <i>Other...</i> 2. In the dialog that appears select <i>Portal Application</i> → <i>Create a new Portal Application Object</i>. 3. Select the project to create the service in. Choose <i>Next</i>. 4. Enter the data as required. Choose <i>Finish</i>.
Create a new portal service from a WSDL file	<ol style="list-style-type: none"> 1. From the menu path, choose <i>File</i> → <i>New</i> → <i>Other...</i> 2. In the dialog that appears select <i>Portal Application</i> → <i>Create a new Portal Application Object</i>. 3. Select <i>Portal Web Service</i>. Select the type of Web services you want to create. Choose <i>Next</i>. 4. Select the portal service to convert to portal Web service or point to the location of the <i>WSDL</i> file. Choose <i>Next</i>. 5. Select the methods you want to expose. Choose <i>Next</i>. 6. Enter data as required. Choose <i>Finish</i>.

1.4.3 Managing PAR and JAR Files in the Project

Purpose	Steps
Deploying or exporting a PAR file	<ol style="list-style-type: none"> 1. Choose <i>File</i> → <i>Export</i> → <i>PAR File</i>. Choose <i>Next</i>. 2. Select the project where the PAR file is located. Choose <i>Next</i>. 3. Enter data as required. If you want to deploy the PAR, choose <i>Deploy PAR</i>. 4. Choose <i>Finish</i>.
Importing a PAR file	<ol style="list-style-type: none"> 1. Choose <i>File</i> → <i>Import...</i> → <i>PAR File</i>. 2. Specify the location of the PAR file. 3. Enter data as required. 4. Choose <i>Finish</i>.  <p>The JAR files, located in a PAR file are not imported into the project. If you want to use some JAR files, you have to import these files to the project explicitly.</p>
Adding a JAR file	<ol style="list-style-type: none"> 1. Open the project context menu and choose <i>Properties</i> → <i>Java Build Path</i>. 2. Choose the <i>Libraries</i> tag. 3. Choose <i>Add External JARs...</i> and browse to the location of the file.
Removing a JAR file	<ol style="list-style-type: none"> 1. Open the project context menu and choose <i>Properties</i> → <i>Java Build Path</i>. 2. Choose the <i>Libraries</i> tag. 3. To remove a JAR file, select it and choose the <i>Remove</i>.

1.4.4 Enterprise Portal Unit Test Studio Perspective

Definition

The *Enterprise Portal Unit Test Perspective* is an extension of the PRT test framework that enables you to test portal applications in their real runtime environment and provides XML-based results. This allows you to test portal applications directly from the development environment.

Use

Using *Enterprise Portal Unit Test Perspective* you can:

- Perform regression tests.
You can compare the test result with a *reference* test result. Note that a *reference* test result is a regular test result with no specific format.
- The preferences of the perspective enable you to specify and save the configuration of the servers containing the tests and the name of the folder for the result, test suite and reference.
- Compare the test result with the reference.



The output of the PRT test framework is XML-based.

- Extract XML output on your computer as a test result.
The portal can contain many testable portal applications.
- Include or exclude tests and save the selection as a test suite.



Not all portal applications can be tested. An application must be suitable for testing. That means that in the *portalapp.xml* the *Testable* property must be set to *true*. For more information, see [Application Configuration \[Page 9\]](#).

See also:

[Writing Test Code \[Page 33\]](#)

[Unit Test Studio Helper Classes \[Page 33\]](#)

[Test Example \[Page 33\]](#)

1.4.4.1 Writing Test Code

A portal component providing testing code must implement the *ITestable* interface and implement the `getTestName()` method.



```
public class MyAbstractPortalComponent
    extends AbstractPortalComponent
    implements ITestable {

    public String getTestName() {
        return "My test case";
    }

    ...
}
```

You can code several test case methods in a single portal component. All test methods must stick to the following pattern (where XXX can be any string supported by Java):



```
public void testXXX(
    IPortalComponentRequest request,
    IPortalComponentTestResponse response) {
    ...
}
```

Example



```
public void testMode(
    IPortalComponentRequest request,
    IPortalComponentTestResponse response) {
    // ... CODING GOES HERE ...
    INode node = request.getNode();
    NodeMode nodeMode = node.getNodeMode();
    response.assert(nodeMode != INode.EDIT, "EDIT Mode not set!",
        "");
}
```

The method expects two parameters:

- A regular implementation of *IPortalComponentRequest*
- A response enriched with test-specific methods.

IPortalComponentTestResponse interface:

```
public interface IPortalComponentTestResponse {
    public void assert(boolean condition, String msgLog, String
msgId);
    public void verify(boolean status, String msgLog, String
msgId);
    public void verify(
        boolean status,
        String msgLog,
        Exception e,
        String msgId);
    public void log(String msgLog, String msgId);
}
```

The `assert()` method checks the condition. If the condition is *false* the test case will show up as an error in the test result. The `assert()` method interrupts the test case at the first error.

The `verify()` method does the same but does not interrupt the test case. This allows you to put several checks in a single test case.



```
public void testMode(
    IPortalComponentRequest request,
    IPortalComponentTestResponse response) {
    //...CODING GOES HERE...
    INode node = request.getNode();
    Response.verify(node != null, "Node is null", "");

    NodeMode nodeMode = node.getNodeMode();
    response.assert(nodeMode != INode.EDIT, "EDIT Mode not set!",
        "");
}
```

}

You can use the `log()` method to write additional information to the `PortalComponentTestResponse`. This information will be included in the test result.

Source Compilation Information

The PRT Test Framework is an extension of the PRT API. The whole Test Framework API is delivered in the `prtttest.jar` library.

Impact on Performance

It is important to understand that you can enrich the portal application with test cases very easily and without any impact on performance. The test cases are executed only when the portal component is invoked using the test mode. This information is provided using specific URL parameters.

1.4.4.2 Unit Test Studio Helper Classes

Definition

The *Unit Test Studio* provides additional helper classes to help with coding more complex *Test Cases*. These helper classes are delivered in the `unittest.facilities.jar` part of the `unittest.facilities.par` portal application. The main class is `com.sapportals.portal.prt.unittest.AbstractTestComponent`. This class contains methods to facilitate the creation of new tests.

A new test must extend `AbstractTestComponent` class instead of extending `AbstractPortalComponent` and implementing `ITestable`. The `AbstractTestComponent` includes two inner classes:

- `TestResult`

The class has attributes corresponding to a result: `boolean` and `log`.

- `TestResultSet`

The class `TestResultSet` is a list of `TestResult`.

`AbstractTestComponent` allows another component to be called thanks to the `invoke()` method. The last parameter, a string, is the URI of the component.

The `displayTestResultSet()` method allows the list of a result to be displayed in the response. Each result is displayed as by the `verify()` method of the `IPortalComponentTestResponse`.

1.4.4.3 Test Example

Example Test One

```
public class TestModes extends AbstractTestComponent {

    public static final String EDIT = "edit";

    public void testModeNode(
        IPortalComponentRequest request,
        IPortalComponentTestResponse response) {
```

```
String IMPL_CONTEXT_NAME =
    request.getComponentContext().getComponentName() +
    "Impl";

AbstractTestComponent.TestResultSet resultSet = new
TestResultSet();

    IPortalComponentURI uri =
request.createPortalComponentURI();

    //Test mode EDIT
    uri.setContextName(IMPL_CONTEXT_NAME);
    uri.setNodeMode(NodeMode.EDIT_MODE);

    invoke(resultSet, request, response, uri.toString());
    list = (ArrayList) resultSet.getData();
    result =
        new TestResult(EDIT.equals((String) list.get(0)),
        "Check do Edit");
    resultSet.addTestResult(result);

    if (EDIT.equals((String) list.get(0)) == true) {
        //check Mode persistency
        String genURI = (String) list.get(1);
        invoke(resultSet, request, response, genURI);
        list = (ArrayList) resultSet.getData();

        result =
            new TestResult(
                EDIT.equals((String) list.get(0)),
                "Check persistency of mode EDIT");

        resultSet.addTestResult(result);
    }
    displayTestResultSet(resultSet, response);
}
```

Example Test Two

The invoke method allows the *TestModesImpl* component to be called in edit mode: the `doEdit()` method is called:

```
public class TestModesImpl extends AbstractPortalComponent {

    protected void doEdit(
        IPortalComponentRequest request,
        IPortalComponentResponse response) {
        System.out.println(">>>>>>>>>>>>DO EDIT IS CALLED");

        AbstractTestComponent.TestResultSet result =
            (AbstractTestComponent.TestResultSet) request
                .getServletRequest()
                .getAttribute(
                    AbstractTestComponent.UNIT_TEST_RESULTSET);
        if (result != null) {
            ArrayList list = new ArrayList();
            list.add(TestModes.EDIT);
```

```

        IPortalComponentURI uri =
request.createPortalComponentURI();
        list.add(uri.toString());
        result.setData(list);
    }
}

```

1.4.5 Enterprise Portal Web Services Checker Description

Definition

The *Enterprise Portal Web Services Checker* view is a part of the *Enterprise Portals Perspective*.

Use

You can use the functionality to query and test existing Web services. You do this by providing the plug-in with an existing WSDL file. You can:

- Create a new tab containing a new SOAP view.
- Close the current SOAP view.
- Retrieve the content of the WSDL file specified in the *WSDL URL* field.
- Send the SOAP request currently visible in the *SOAP Request* pane to the Web service endpoint.
- Start your Internet browser and point it to the *SOAP Administration* page of your portal installation.
- Start your Internet browser and point it to the *SOAP log viewer* page of your portal installation.
- Save the content of all fields in an XML file.

This perspective allows you to:

Function	Procedure
Retrieve a WSDL file	<ol style="list-style-type: none"> 1. Enter the URL of the file in the <i>WSDL URL</i> field. 2. From the menu choose <i>Get WSDL content</i> icon.
Create a SOAP request for an operation	<ol style="list-style-type: none"> 1. Choose the Request sub-tab. 2. Select the operation and open the context menu. 3. Choose <i>Cerate SAOP Request</i>.
Use the <i>SOAP Request</i> sub tab	Here you can browse and change the content of a SOAP request
Use the <i>XML</i> sub tab	Allows you to view the XML Source of the WSDL.
Send an SOAP request	Choose the <i>Send Message</i> icon.

Use the <i>SOAP Response</i> sub tab	Here you can browse the content in a tree, display the XML source of the response or the HTTP headers of the response.
--------------------------------------	--

1.4.6 Development Configuration

Use

A development configuration consists of an XML file, which describes the different servers to be used for NetWeaver Development Infrastructure development:

- Design Time Repository (DTR) server
- Landscape Directory server
- Component Build Server (CBS)

Use this procedure to create a new project of type *Portal Application Development Component*. You can build either standalone or application module development components.

Enterprise Portal development is split into the following parts:

- Portal Application Module

Contains the model classes and structures corresponding to the *Enterprise Portal Application* project.

When you create a *Portal Application Module* development component, the SAP NetWeaver Developer Studio automatically creates a public part for assembly containing the PAR archive, and a public part for compilation containing the API part. You should not modify public parts that were generated automatically. The build result of this development component is compiled classes and a PAR file.

- Portal Application Standalone

A *Portal Application Standalone* development component contains the model classes and structures corresponding to the *Enterprise Portal Application* project. The difference from the *Portal Application Module* is that these projects are automatically built in a deployable archive (SDA file).

When you create a *Portal Application Standalone* development component, the SAP NetWeaver Developer Studio automatically creates a public part for assembly containing the PAR archive, and a public part for compilation containing the API part. The build result is compiled classes and SDA (containing PAR) that are ready to be deployed on the server



Do not modify public parts that were generated automatically.

See also:

- [Managing Development Components \[Page 33\]](#)
- [Creating References Between PAR Development Components \[Page 33\]](#)
- [Package PAR Development Components in an EAR file \[Page 33\]](#)

1.4.6.1 Managing Development Components

Procedure

If You Want To	Then
Create a new development component	<ol style="list-style-type: none"> 1. From the menu path, choose <i>File → New → Project</i>. 2. Select <i>Development Component → Development Component Project</i> and choose <i>Next</i>. 3. Select the <i>Software Component</i> from the list where you want to add the development component. Choose <i>Next</i>. 4. Specify the type of the development component. <ul style="list-style-type: none"> ○ To build a standalone application, under <i>Type</i> choose <i>Enterprise Portal → Portal Application Standalone (Packed in SDA)</i>. Choose <i>Next</i>. ○ To build an application module development component, under <i>Type</i> choose <i>Enterprise Portal → Portal Application Module</i>. Choose <i>Next</i>. 5. Choose <i>Create an Empty Project</i> and specify the <i>archive name</i>. Set the deployment type to <i>EAR SDA</i>. Choose <i>Finish</i>.
Create references between PAR development components	See Creating References Between PAR Development Components [Page 33]

Import a Portal Application DC project from DTR	<ol style="list-style-type: none"> 1. Log on to the DTR system Under the <i>Design Time Repository</i> perspective, choose the <i>Log on...</i> icon. 2. Under the <i>Development Configurations</i> perspective, open the <i>Inactive DCs</i> view. 3. Explore the content of <i>Development Configuration</i> and identify the portal development component you created in DTR. 4. Open the context menu for this development component and choose <i>Create project</i>. 5. Select the development components you want to sync and choose <i>OK</i>.
Check in an activity	<ol style="list-style-type: none"> 1. Open the <i>DTR</i> perspective. 2. Go to the <i>Open Activities</i> view. 3. Open the activity context menu and choose <i>Check in</i>.
Package PAR development components in an EAR file	See Package PAR Development Components in an EAR file [Page 33]

1.4.6.2 Creating References Between PAR Development Components

Procedure

1. Create a new *Portal Application Module DC Project* named *tc/epbc/test/usingdc* with par name *com.sap.test.usingdc.par*
2. In *Portal DC Explorer*, expand the metadata node of this component to show the *Used DCs*.
3. Open the context menu for the *Used DCs* and select *Add Used DC...*
4. Expand the *Development Configuration* to display the public part of the *Portal Application DC tc/epbc/test/useddc*. Select the *API* public part of this development component.
5. Select the following boxes: *BuildTime*, *DeployTime*, *RunTime*.
6. Choose *Finish*.
7. Rebuild the development component project using the menu command *Project → Rebuild DC Project*.
8. Create a new *Portal Component* that calls the *TimeService* as follows:



```
public void doContent(  
    IPortalComponentRequest request,
```

```

        IPortalComponentResponse response) {
        response.write(
            "Call to time com.sap.test.useddc from
com.sap.test.usingdc:");
        response.write("<br>");
        ITimeSvc timeSvc =
            (ITimeSvc)
PortalRuntime.getRuntimeResources().getService(
            "com.sap.test.useddc.timeService");
        response.write(timeSvc.getDate());
    }

```

9. Open the *portalapp.xml* and add *SharingReference* property and set its value to *com.sap.test.useddc*.
10. Create and deploy the PAR file on the portal.
11. Test the components you have created by opening the *portalapp.xml* file of the project.
12. Switch to the *Overview* tab and click run on the *TimeComponent* component.

1.4.6.3 Package PAR Development Components in an EAR file

13. Create a new development component of type *Enterprise Portal* → *Portal Application Module* in *Local Development* → *My Components*. Name this development component *tc/epbc/test/parinear1*. Name the PAR file *com.sap.test.parInEar.par*.
14. Create a new portal component of type *AbstractPortalComponent* in the portal development component. Call this component *testComponent*. Add a *Hello World* to *doContent* method as follows:



```

public void doContent(
    IPortalComponentRequest request,
    IPortalComponentResponse response) {
    response.write("TestAC2: Hello World!")
}

```

15. Create a new development component of type *J2EE* → *Enterprise Application* in *Local Development*. Name this development component *tc/j2ee/test/ear1*.
16. In the *J2EE DC Explorer* view, expand the development component *metadata/DC Definition* nodes of this *J2EE* development component to display *Used DC's*.
17. Open the context menu on *Add Used DC*.
18. Expand *Local Development* → *My Components* and locate *tc/epbc/test/parinear1*. Expand the development component definition, open public part node and select the *PAR Public Part* of the development component to which you want to create a reference.
19. Make sure only the *BuildTime* option is selected and choose *Finish*.
20. From the *J2EE DC Explorer* view, open the *application-j2ee-engine.xml* of *LocalDevelopment~tc~j2ee~test~ear1 J2EE DC* project.
21. Go to the *expert setting* tab to display the module node. In the *Module to deploy* field, type the PAR name of the portal development component: *com.sap.test.parInEar.par*. Expand this new node to show *specify container* and type: *PortalRuntimeContainer*. Save the file.

22. Switch to *Source* tab to check the content of the XML file.
23. Make a *DC Build* of this *J2EE Enterprise Application* development component.
24. In the *J2EE DC Explorer* view, open the context menu for the *LocalDevelopment~tc~j2ee~test~ear1* project and choose *Deploy*.

1.4.7 Registering an Additional Plug-in

Use

You can use an API that enables you to register a plug-in. Such a plug-in can be called during the generation of a PAR file. This enables you to extend the original portal application development model.

You can interfere with both processes concerning the packing and unpacking of the PAR file.

- During the packing operation, the additional tool can be called before the final packing of the PAR file.
- During the unpacking operation, the additional tool is called after the initial unpacking of the PAR file.

Prerequisites

The new plug-in must:

- Be loaded at SAP NetWeaver Developer Studio startup.
- Be registered to the PAR plug-in.
- Implement two interfaces, one for the packing and another for the unpacking.

Procedure

Implementation

Implement the *org.eclipse.ui.Istartup* interface with its method *earlyStartup()* in the base class of the add-on plug-in (the first class that is called when the add-on plug-in is started, and where *org.eclipse.ui.plugin.AbstractUIPlugin* is extended).

In the *plugin.xml* add:



```
<extension
    point="org.eclipse.ui.startup">
</extension>
```

Register

Call the static method of *com.sap.portal.developmentTools.ideSpecific.eclipse.PortalPlugin* class *registerPlugin(String pluginId, IParIdePlugin aPlugin)*. The plug-in ID must be unique.

Interfaces

To register an add-on plug-in, you must implement the `com.sap.portal.developmentTools.general.api.IparIdePlugin` and set the implementation class as a parameter when it registers.

The add-on plug-in has to specify the final state of the operation clearly using the method `setFinalState(int status)` from the class `com.sap.portal.developmentTools.general.api.IfinishResult`. If the final state is *OK* or *Warning* then the PAR process continues; otherwise, it is stopped. Inside the *IfinishResult*, you can add messages with their level.

Result

These will be displayed in the task view of SAP NetWeaver Developer Studio.

2 Go and Create

This section describes how to quickly start writing portal applications, and includes the following:

- [Creating Your First Portal Application \[Page 33\]](#)

2.1 Creating Your First Portal Application

Purpose

The Portal Runtime Technology (PRT) manages portal applications and portal services. See document [Portal Runtime Basics \[Page 33\]](#) for programming details of the PRT.

This document describes the principles of a JSP DynPage and provides a basic example with JSP and describes which methods have to be implemented. The next step is the event handling of a JSP DynPage and the data exchange between the JSP DynPage and the JSP.

To work with this document you need a basic understanding of Java Server Pages (JSP). Sun provides JSP documentation. The Portal Runtime (PRT) has made modifications to the JSP standard. For details see Java Server Pages (JSP) Support in the Portal Runtime (PRT).

The example has following steps:

[Creating the JSPDynPage \[Page 33\]](#)

[JSPDynPage event handling \[Page 33\]](#)

[Data exchange between JSPDynPage and JSP \[Page 33\]](#)

[Data Exchange Using a Bean \(used in the example\) \[Page 33\]](#).

Alternative: [Data Exchange Using the Session Object \[Page 33\]](#)

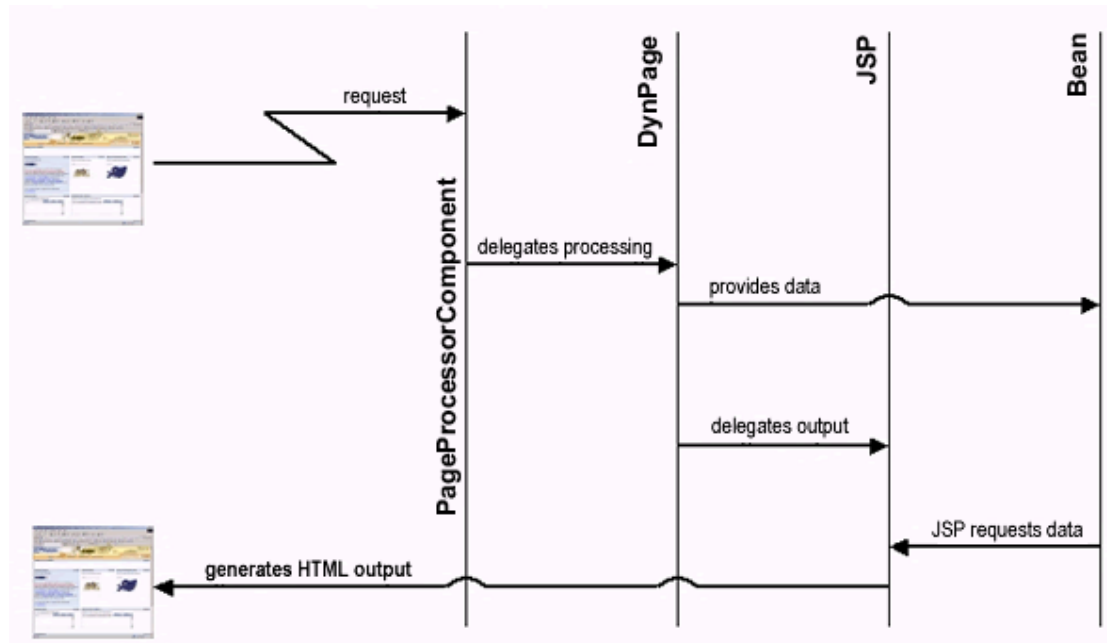
Alternative: [Data Exchange Using the Context Object \[Page 33\]](#)

Alternative: [Data Exchange Using the Request Object \[Page 33\]](#)

Concept of the JSP DynPage

JSP/Servlets offer basic event handling, you have to take care of the event handling yourself (analysing the received form, getting the sender of the event etc.). In addition, the programmer has to take care of the session identifier, which is a unique identifier that makes sure that the datasets are user specific. The JSP DynPage provides enhanced event handling and easy session management. In this example we use the HTML-Business for Java (HTMLB) controls to create the Graphical User Interface (GUI). HTMLB is a Portal service.

Dataflow of a DynPage Component



2.1.1 Portal Runtime Basics

Purpose

The Portal Runtime Technology (PRT) defines and manages two kinds of objects that define a portal application:

- portal components
- portal services

From a user point of view, a portal component can be displayed into a portal page. From an application development perspective, the portal component is a pluggable component that is designed to be executed into a portal environment.

Creating Your First Portal Application

A portal component has an application part (for example, a Java class `IPortalComponent` or `AbstractPortalComponent` and/or Java Server Pages (JSP)) and can use resources (such as images and language bundles).

To deploy the portal component into the portal you have to build a portal archive. All files that belong to the portal component are packaged in this portal archive in a specific structure. You can even package more than one portal application in a portal archive. One specific file the Deployment Descriptor plays a major role during deployment and runtime.

A portal service has to implement the `IService` interface. Refer to portal service implementation for further details.

To build portal applications the PDK provides an IDE support for Eclipse. It also provides an ANT script to build the par files.

This description of the PRT is a short introduction. For the complete PRT guide refer to the Portal Runtime Technology tab in the top level navigation or use the PDF version of the PRT guide.

IPortalComponent

This example shows how you can implement `IPortalComponent`, the central abstraction of the Portal Component API. It is not recommended to implement this interface directly. But for a deeper understanding of the runtime, it makes sense to take a look on it.

During the portal component life cycle the runtime calls the following methods of

`IPortalComponent`:

- `init()`
Is called when the portal component is initialized. The runtime always initializes one instance of the portal component. This instance is shared among different users and sessions
- `service()`
Is called when a client request for this portal component is received. All rendering is done in this method.
- `destroy()`:
Is called when the portal component is discarded from memory. This can happen either because the runtime is terminating or because the runtime needs to free up some memory or during uploading of a new version of the portal component.

The most important method is `service(..)`. Here the portal component output is written to the client and events are handled.

Output is created with the `write()` method of the response object, which is available as a method parameter.

Creating Your First Portal Application

```
public void service(IPortalComponentRequest request,
    IPortalComponentResponse response) throws PortalComponentException {
    System.out.println("service");
    // dispatch the events
    IPortalRequestEvent event = request.getRequestEvent();
    //event handling
    if (event!=null){
        String eventName = event.getName();
        if (eventName.equalsIgnoreCase("compute"))
            doCompute(request,event);
        if (eventName.equalsIgnoreCase("back"))
            doBack(request,event);
    }
    // call the content creation method
    doContent(request,response);
}
```

You can put your output into the response with method write().

```
response.write("Hello World");
```

IPortalComponentRequest

The `IPortalComponentRequest` contains request specific data and provides access to the environment in which the `PortalComponent` is running. The following Objects are accessible through the `IPortalComponentRequest`:

- `ServletConfig`, `HttpServletRequest`, `HttpServletResponse`: Those objects are the original servlet objects.
- `IPortalComponentSession`, `IUserContext`, `Locale`: Those objects provide user specific information
- `IPortalComponentContext`: This is the context in which the portal component is executed
- `IResource`: Is a representation of an external resource (like an image, css, ...)
- `ILogger`: To be used to write information to the portal log
- `IPortalComponentProfile`: The profile of the portal component

In addition, the `IPortalComponentRequest` acts as a factory for URLs like the component URI or request events.

The way you can access the `IPortalComponentResponse` depends how you are implementing your portal component:

- `IPortalComponent`: as an argument of method `service()`
- `AbstractPortalComponent`: as an argument of method `doContent()` and all other standard event handler methods (like `doEdit()`)

IPortalComponentResponse

The `IPortalComponentResponse` is used for creating the output to the client. The `IPortalComponentResponse` object provides the following methods:

- `addCookies()`: You can add a cookie to your response
- `write()`: You can directly write strings to the response. Usually, you will write HTML code with this method.
- `include()`: You can include the output of another portal component or a portal component resource like a JSP, an applet, a CSS, ...

The way you can access the `IPortalComponentResponse` depends on how you are implementing your portal component:

- `IPortalComponent`: as an argument of method `service()` .
- `AbstractPortalComponent`: as an argument of method `doContent()` and all other standard event handler methods (like `doEdit()`)

AbstractPortalComponent

Create Output

When you implement the `AbstractPortalComponent` class you have to extend the `doContent(IPortalComponentRequest request, IPortalComponentResponse response)` method. In the `doContent()` method you can output a string with the `write` method of the response object:

```
response.write("Hello World");
```

Event Handling

First step is to create the event that is raised when the user clicks, for example, a button. The portal runtime provides the interface `IPortalComponentURI` to generate this events:

```
// create a URI for the event
uri = request.createPortalComponentURI();
uri.setPortalRequestEvent(request.createRequestEvent("compute"));
```

To create an URL that can be used, for example, in a HTML, you have to convert the uri into an url string:

```
uri.toString();
```

If the user clicks on the button, the portal runtime will dispatch this event to a method named `doCompute (do<Eventname>)` . You have to implement this method in your `AbstractPortalComponent`. The `IPortalRequestEvent` object contains the attributes of the event.

```

public void doCompute(IPortalComponentRequest request, IPortalRequestEvent event) {
    try {
        name = event.getData().getAttribute("expression");
        state = WELCOME_STATE;
    } catch (NullPointerException npe) {
    }
}

```

2.1.2 Creating the JSPDynPage

First step to create a portal component is to define a class that works as loader class - it inherits from the `PageProcessorComponent`. The created loader class (in the following example named `ExampleOneDyn`) executes the method `getPage()` and returns a unique value of the JSP DynPage we can use (in the following example named `DynPageOne`).

```

package com.mycompany.basicexample;

import com.sap.htmlb.page.DynPage;
import com.sap.portal.htmlb.page.PageProcessorComponent;

public class ExampleOneDyn extends PageProcessorComponent {

    public DynPage getPage() { // Has to be overridden
        // Calls the DynPage and returns its value as DynPageOne
        return new DynPageOne();
    }
}

```



The SAP EP Developer Plug-ins provide a JSP Dynpage wizard, that creates all necessary classes, beans and JSP for a JSP Dynpage.

The class `DynPageOne` is extended from the `DynPage` class. Following methods have to be overwritten:

- `doInitialization`

Called when the application is started. The call is made when the page is directly called per URI without parameters and no event occurred.

Usually this method is used to initialize data and to set up models. Be aware of the fact that the `doInitialization` event is also caused when another portal component on the same page sends an event.



With the "Personalize" Dialog you can compose a page by grouping several portal components together. We have created a page called `myPage` with two portal components - A and B. When calling the page `myPage` the `doInitialization` is called from portal component A and B followed by the call of the method `doProcessBeforeOutput`. When an event occurs in the portal component B (for example, by clicking on a button), the `doInitialization` method in portal component A is called again, while in portal component B the method `doProcessAfterInput` followed by the event handling method assigned for the button and finally the `doProcessBeforeOutput` method.

Creating Your First Portal Application

To create solid portal components you must be aware of the fact and check in the `doInitialization` method if the data has already been initialized or the models have been created. Otherwise your portal component is always reset to the initial state if an event in another portal component occurs.

The Enterprise Portal treats every portal component isolated. In this case an event in one portal component does not cause the `doInitialization` event in the other portal component on the same page. The PDK can emulate this behavior by setting the `ISOLATED` flag in the page description XML file to true. The page description XML file in the content folder of the page and defines the position of the portal component, height and tray type.



Example for an entry in the XML file:

```
<component name="AAA.default" title="AAA.default"
height="400" trayType="SAPTrayD3" Position="1"/>
```

Example for an entry in the XML file with isolation flag set so that PDK behaves like the Enterprise Portal:

```
<component name="AAA.default" title="AAA.default"
isolated="true" height="400" trayType="SAPTrayD3"
Position="1"/>
```

If you use the Page Editor in the DevTools section of the PDK you can set the isolated flag interactively.

- **doProcessAfterInput**
Called when the web client sends the form to the web server. Except on `doInitialization` (see above) the call is performed every time an event occurs.
- **doProcessBeforeOutput**
Called before the form is sent to the web client. The call is performed every time even on `doInitialization`.

In our example we only use the `doProcessBeforeOutput` method, the other methods stay "empty".

Creating Your First Portal Application

```

package com.mycompany.basicexample;
import com.sap.htmlb.*;
import com.sap.htmlb.enum.*;
import com.sap.htmlb.page.PageException;
import com.sap.portal.htmlb.page.JSPDynPage;

public class DynPageOne extends JSPDynPage {

    /* Constructor */
    public DynPageOne() {
        this.setTitle("DynPageOne");
    }

    /* Used for user initialization. Called when the application is
    * started */
    public void doInitialization() {
    }

    /*
    * Used for handling the input. Generally called each time when an
    * event occurs on the client side.
    */
    public void doProcessAfterInput() throws PageException {
    }

    /* Used for handling the output. This method is always called.
    In our example the JSP makes a textView that displays
    "May the force be with you unknown user". */

    public void doProcessBeforeOutput() throws PageException {
        // set the JSP which builds the GUI
        this.setJspName("OutputText.jsp");
    }
}

```

JSP - OutputText.jsp - that is called by doProcessBeforeOutput

```

<!-- OutputText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
    id="myContext">
    <hbj:page
        title="An Easy Start">
        <hbj:form>
            <hbj:textView
                id="welcome_message"
                text="May the force be with you unknown user"
                design="HEADER1"
            />
        </hbj:form>
    </hbj:page>
</hbj:content>

```

Deployment descriptor: Necessary entries to execute this portal component:

Creating Your First Portal Application

```

<application>
  <application-config>
    <property
      name="SharingReference"
      value="htmlb"/>
    </application-config>
  <components>
    <component
      name="default">
      <component-config>
        <property
          name="ClassName"
          value="com.mycompany.basicexample.DynPageOne"/>
        <property
          name="SecurityZone"
          value="com.sap.pct.pdk/low_safety"/>
        </component-config>
        <component-profile>
          <property
            name="tagLib"
            value="/SERVICE/htmlb/taglib/htmlb.tld"/>
          </component-profile>
        </component>
      </components>
    <services/>
  </application>

```

The entry for `ClassName` is case sensitive. The entry `SharingReference` makes sure, that the necessary HTMLB libraries are found.

The strength of the JSP `DynPage` is the event handling. The JSP `DynPage` follows the concept of Java controls (for example, Swing) - Java controls, like the HTMLB controls, can have one or more events. You define the event by assigning a method name to the event. The method is called whenever the event is raised (for example, when a button is clicked). The event handling method is coded in the JSP `DynPage`. The JSP `DynPage` does the event handling and calls the proper event handling method.

Next step in our example is to place a button to our user interface and define an event for it.

2.1.3 JSPDynPage Event Handling

Some HTML-Business for Java controls have an event attribute (for example, see button). The button for example has an 'onClick' attribute that specifies the name of the method which should handle the event. The event will occur when the appropriate user action takes place - in this case clicks on the button. The name of the method specified with the 'onClick' attribute has to be declared in the JSP `DynPage`.

HTML-Business for Java: Statement to specify the method name:

```
Button1.setOnClick("myClick");
```

JSP `DynPage`: Declaration of the method that processes the event:

Creating Your First Portal Application

```
public void myClick (Event event) { ..coding.. }
```

or

```
public void onMyClick (Event event) { ..coding.. }
```

Both declarations are valid. The decision to use the on... method declaration could be helpful to make it obvious that this method handles an event.



With the on.. declaration method the first letter of the declared event name must be a capital letter.

If both methods are implemented (myClick and onMyClick), only the method myClick, will be called. The method onMyClick will be ignored.

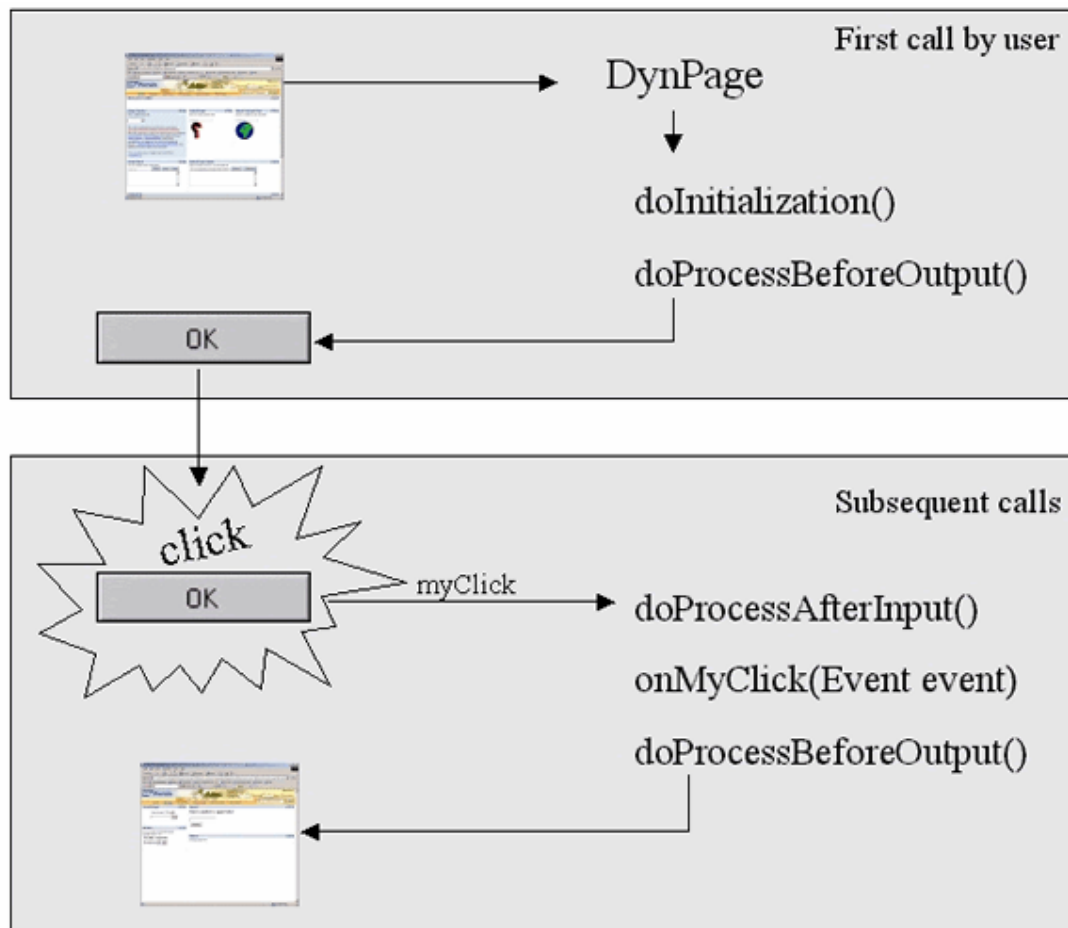


Some HTML-Business for Java controls have a 'setOnClient' attribute. With this attribute you specify a JavaScript fragment that handles the event on the Web client side. The event is NOT transmitted to the Web server.

If an event occurs it is handled as follows (doInitialization is performed when the application starts).

- doProcessAfterInput
Called when the web client sends the form to the web server.
- onMyClick
The event handling method we declared.
- doProcessBeforeOutput
Called before the form is sent to the web client.

Event Processing



2.1.4 Data Exchange between JSPDynPage and JSP

The storing methods we discuss in this section are volatile in the sense that the data is lost when the session is over (or even before that). Generally you have to decide if the data you provide in the JSP DynPage should be shared among other users and how long the data must "live". For storing data permanently you can refer to the Profile documentation.

Storing data can be performed using:

- beans

A bean is defined with set and get methods. The bean can be accessed from the JSP DynPage and the JSP as well as from other users (depending on the scope, see "Usage of Beans").

- session

Data stored in the http session will be kept by the server as long as the user session is alive.

- context

Creating Your First Portal Application

The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources.

- request

Data stored in the request are kept for the request.

2.1.4.1 Data Exchange Using a Bean

A bean is used to get and set "dynamic" data. The JSP DynPage usually provides the bean with data and the JSP reads the data. The functionality of the basic example is extended by an input field that allows user input. The user input is stored in a bean and then displayed as text by a JSP program.

Following steps are necessary

- create a bean
- initialize the bean
- introduce the bean to the JSP program `OutputText.jsp`

Declaring a bean in a JSP

The tag [usebean \[Page 33\]](#) declares a bean in a JSP.

- class
Class name of the bean.
- id
Identification name of the bean. The id is used to access the bean in scriptlets.
- scope
Defines the availability of the bean. Details see "[How to use Beans \[Page 33\]](#)".

Attributes	M	Values	Usage – JSP Taglib
class	*	String (cs)	<code>class="com.sap.htmlb.beandemo.myBean"</code>
id	*	String (cs)	<code>id="idOfMyBean"</code>
scope	*	APPLICATION SESSION REQUEST PAGE	<code>scope="APPLICATION"</code>

Bean for the JSPDynPage Example

```
package bean;
/*
 * A simple bean whose only purpose is to store a String.
 * It as a get and set method to store and recall the string.
 */
public class DynPageNameBean {
    public String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

The GUI is extended by an inputField and a button to allow the user to enter a string (for example, user name) and submit the form by clicking the button or pressing Return/Enter on the keyboard.

Following changes to OutputText.jsp are necessary:

- Adding a form to allow the definition of a default button.
- Introduce the bean to the JSP program (<jsp:usebean .../>).
- Changing the textView so that it displays the string retrieved from the bean.
- Adding a label telling the user to enter the user name. The label should start in a new line and one line separated from the textView. That is why you find a

 before the label
- Adding the inputField "user_name_input" - the JSP DynPage retrieves the data in the input field using getComponentByName.
- Adding a send button which we also define as the default button. This enables the user to send his input back to the server by either clicking on the button or by simply pressing Enter/Return on the keyboard when he finished the input.

```
<!-- OutputText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
    id="myContext">
    <hbj:page
        title="An Easy Start">
        <hbj:form
            id="myFormId">
            <!-- Declaration of the bean. -->
            <jsp:useBean
                id="UserNameBean"
                scope="application"
                class="bean.DynPageNameBean"
            />
            <hbj:textView
                id="welcome_message"
                design="HEADER1">
            <%
                welcome_message.setText
                ("May the force be with you "+UserNameBean.getName());
```

```

%>

</hbj:textView>
<br>
<br>
<hbj:label
    id="label_input"
    text="Your name please"
    design="LABEL"
    required="TRUE"
    labelFor="user_name_input"
/>

<!-- inputfield to allow userInput - the inputfield has the id --%>
<!-- "user name input" which is used in the JSP DynPage to --%>
<!-- access the input field and retrieve the input of the user --%>
<hbj:inputField
    id="user_name_input"
    type="STRING"
    design="STANDARD"
    width="250"
    maxlength="30"
/>
<hbj:button
    id="Send_Button"
    text="Send"
    tooltip="Sends my name"
    onClick="onSendButtonClicked"
    width="100"
    design="EMPHASIZED">
<%
    myFormId.setDefaultButton(Send_Button);
%>

</hbj:button>
</hbj:form>
</hbj:page>
</hbj:content>

```

Result

Hint: The default string "unknown user" will be set in our JSP DynPage in the following section.

May the force be with you unknown user

Your name please*

Changes in the JSPDynPage

The JSP DynPage has to be adjusted as well. Following steps are necessary:

- Introducing the bean to the JSP DynPage (Import statement).
- In the doInitialization() method we set a default user name to "unknown user".

Creating Your First Portal Application

- Creating the event method `onSendButtonClicked()` for the button click in which the status (variable state) is set to `WELCOME_STATE` so that the `doProcessBeforeOutput` selects another JSP file.
- In `doProcessAfterInput()` method (which is called whenever an event occurs) we request the inputField "user_name_input" from the JSP by using `getComponentByName` to have access to the user input in the JSP DynPage. If the inputField "user_name_input" is not empty the string is stored in the bean.

```
package com.mycompany.basicexample;
/** introduce the bean */
import bean.DynPageNameBean;
import com.sap.htmlb.*;
import com.sap.htmlb.enum.*;
import com.sap.htmlb.event.Event;
import com.sap.htmlb.page.DynPage;
import com.sap.htmlb.page.PageException;
import com.sap.portal.htmlb.page.JSPDynPage;
import com.sap.portal.htmlb.page.PageProcessorComponent;
import com.sap.portal.prt.component.IPortalComponentContext;
import com.sap.portal.prt.component.IPortalComponentProfile;
import com.sap.portal.prt.component.IPortalComponentRequest;

public class DynPageOne extends JSPDynPage {
    private final static int INITIAL_STATE = 0;
    private final static int WELCOME_STATE = 1;
    private int state = INITIAL_STATE;
    private String name;

    /**
     * Constructor
     */
    public DynPageOne() {
        this.setTitle("Become a Jedi");
    }

    /**
     * Used for user initialization. called when the application is
     * started
     */
    public void doInitialization() {
        // create the bean and set a default text value "unknown user"
        IPortalComponentRequest request =
            (IPortalComponentRequest) this.getRequest();
        IPortalComponentContext myContext = request.getComponentContext();
        IPortalComponentProfile myProfile = myContext.getProfile();
        // new bean object
        UserNameContainer = new DynPageNameBean();
        // set default name
        UserNameContainer.setName("unknown user");
        // store bean in profile for the JSP
        myProfile.putValue("UserNameBean", UserNameContainer);
        // Set the state so that we can decide what action to do next
        state = INITIAL_STATE;
    }

    /**
     * Used for handling the input. Generally called on each event
     * we use this method to get the user name and store it in the bean
     */
}
```


Creating Your First Portal Application

```

public void doProcessAfterInput() throws PageException {
    // get the input field from the JSP
    InputField myInputField =
        (InputField) getComponentByName("user_name_input");
    if (myInputField != null) {
        this.name = myInputField.getValueAsDataType().toString();
    }
    IPortalComponentRequest request =
        (IPortalComponentRequest) this.getRequest();
    IPortalComponentContext myContext = request.getComponentContext();
    IPortalComponentProfile myProfile = myContext.getProfile();
    DynPageNameBean myNameContainer =
        (DynPageNameBean) myProfile.getValue("MyNameBean");
    myNameContainer.setName(name);
}
/**
 * Used for handling the output. This method is always called.
 * In our example before the JSP made a textView with
 * "May the force be with you unknown user".
 * We now extend this method that according to the state it either -
 * that is when state = INITIAL_STATE - asks for the user name
 * and calls the user "unknown user" or after init - that is when
 * state = WELCOME_STATE - displays a success message with
 * the username.
 */
public void doProcessBeforeOutput() throws PageException {
    switch (state) {
        case WELCOME_STATE :
            this.setJspName("OutputSuccessText.jsp");
    }
    break;
    default :
        this.setJspName("OutputText.jsp");
        break;
}
/**
 * this method handles the event of the button. The event is fired
 * either when the user clicks on the button or presses the
 * Return/Enter key when he is in the inputField (since we defined
 * the button as default button). In this method we set the state to
 * WELCOME_STATE so that on the following doProcessBeforeOutput
 * (which is called immediately after this method)
 * a success message is displayed
 */
public void onSendButtonClicked(Event event) throws PageException {
    state = WELCOME_STATE;
}
}

```

The only thing missing now is OutputSuccessText.jsp which should send the personalized message to the user. The usage of the bean has been already shown in OutputText.jsp so OutputSuccessText.jsp is very small and creates a textView with the username in it.

Creating Your First Portal Application

```
<!-- OutputSuccessText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
  id="myContext">
  <hbj:page
    title="Successful processing">
    <jsp:useBean
      id="UserNameBean"
      scope="application"
      class="bean.DynPageNameBean"
    />
    <hbj:textView
      id="success_message"
      design="HEADER1">
      <%
        success_message.setText
          ("The force is with you" + UserNameBean.getName());
      %>
    </hbj:textView>
  </hbj:page>
</hbj:content>
```

Result of the JSP

(assuming that the user responded with "SAP")

The force is with you SAP**2.1.4.2 Data Exchange Using the Session Object**

Data stored in the http session will be kept by the server as long as the user session is alive.

JSPDynPage

Getting the request object:

```
IPortalComponentRequest request =
    (IPortalComponentRequest) this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getComponentSession().putValue("myText",
                                         "Text in the session
context");
```

To get the stored value we have to use the command line:

```
request.getComponentSession().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%
    componentRequest.getComponentSession().putValue("myText",
                                                    "That text is from the
JSP");
%>
```

To get the stored value we have to use the command line:

```
<%

    componentRequest.getComponentSession().getValue("myText").toString
(); %>
```

2.1.4.3 Data Exchange Using the Context Object

The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources. We strongly recommend to refresh the stored data (for example, in the JSP DynPage method `doProcessAfterInput`, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request =
    (IPortalComponentRequest) this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getComponentContext().putValue("myText",
                                       "A short note in the
context");
```

To get the stored value we have to use the command line:

```
request.getComponentContext().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%
    componentRequest.getComponentContext().putValue("myText",
                                                    "From the JSP");
%>
```

Creating Your First Portal Application

```
%>
```

To get the stored value we have to use the command line:

```
<%  
  
componentRequest.getComponentContext().getValue("myText").toString  
();  
  
%>
```

2.1.4.4 Data Exchange Using the Request Object

The lifetime of data stored in the request is limited to the request only. You have to refresh the stored data (for example, in the JSP DynPage method `doProcessAfterInput`, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request =  
    (IPortalComponentRequest) this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getNode().putValue("myText", "A short note in the  
request");
```

To get the stored value we have to use the command line:

```
request.getNode().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%  
    componentRequest.getNode().putValue("myText", "That is from the  
JSP");  
%>
```

To get the stored value we have to use the command line:

```
<%  
    componentRequest.getNode().getValue("myText").toString();  
%>
```

3 Core Development Tasks

This section describes how to accomplish some of the most common and useful tasks for programming in the portal. The tasks are divided into the following sections:

- [Creating and Managing Content \[Page 33\]](#): Tasks for creating content. The tasks involve the creation of HTML that is viewed within an iView, or the creation of iViews, pages and other portal objects within the PCD.
- [Modifying the Desktop and Navigation \[Page 33\]](#): This section describes the following types of tasks:
 - **Desktop**: Tasks for modifying the look and feel of the portal, including customizing the layout and colors of the portal. You can also create different desktops for different sets of users.
 - **Navigation**: Tasks for enabling users to navigate between content in the portal. You can provide navigation defined in pre-defined roles, navigation based on external data and other types of navigation.
- [Connecting to Backend Systems \[Page 33\]](#): Tasks for connecting to back-end systems and applications.
- [Specialities in the Portal \[Page 33\]](#): Describes special portal features that involve several developer tasks.

3.1 Creating and Managing Content

This section describes how to create and manage content, and includes the following sections:

- [Managing iViews and Other PCD Objects \[Page 33\]](#)
- [Working with XML \[Page 33\]](#)
- [Creating Administration Interfaces \[Page 33\]](#)
- [Client-Side Eventing \[Page 33\]](#)
- [Page Builder \[Page 33\]](#)
- [HTML-Business for Java \[Page 33\]](#)
- [User Management Engine \[Page 33\]](#)
- [User Agent Service \[Page 33\]](#)

3.1.1 Managing iViews and Other PCD Objects

This section describes how to create, look up and modify standard PCD objects, also known as semantic objects. Semantic objects enable you to manage common portal objects, such as iViews and pages, via the portal API.

The following sections are included:

- [Architecture \[Page 33\]](#): Background information on semantic objects.

Creating and Managing Content

- [How to Manage Semantic Objects \[Page 33\]](#): Basic APIs for working with all semantic objects.
- [How to Manage iViews \[Page 33\]](#): Additional APIs for working with iViews.
- [How to Manage Pages \[Page 33\]](#): Additional APIs for working with pages.
- [How to Manage Layouts \[Page 33\]](#): Additional APIs for working with layouts.
- [How to Manage Systems \[Page 33\]](#): Additional APIs for working with systems.
- [Essential Information \[Page 33\]](#): The portalapp.xml configuration and the JAR files for working with semantic objects.

3.1.1.1 Architecture

A semantic object is a collection of attributes that represent a portal object, such as an iView or page. Such semantic objects are persisted in the PCD. By creating and modifying these objects in the PCD, you can change the iViews and pages and other portal objects that are displayed in the portal or that are available to administrators.

Each semantic object implements `IAttributeSet` (from the `com.sap.portal.pcm.admin` package), which defines methods for modifying its attributes.

Objects

For each type of semantic object, there are generally two types of Java interfaces:

- The semantic object interface, such as `IiView` for an iView.
- A helper service interface that provides helper methods for working with that type of semantic object, such as `IiViews` for iViews.

Some helper services simply provide special implementations of the methods defined in the `com.sap.portal.pcm.IObjectsManager` interface, which these helper services implement. Others provide additional methods. For example, `IiViews` defines no new methods, while `ISystems` provides additional methods, for example, for retrieving all systems defined in the PCD.

To obtain an instance of the helper service, create an instance of the service with the `PortalRuntime` class, as you would for any service.

```
IiViews iViewSrv = (IiViews)PortalRuntime.getRuntimeResources()
    .getService(IiViews.KEY);
```

The following are the interfaces for key semantic objects:

Semantic Object	Interfaces
iView	<code>IiView</code> , <code>IiViews</code>
Page	<code>IPage</code> , <code>IPages</code>
Layout	<code>ILayout</code> , <code>ILayouts</code>
System	<code>ISystem</code> , <code>ISystems</code>

These objects are part of the `com.sap.portal.pcm` package.

3.1.1.1.1 Atomic Name

Each object in the PCD is specified by its PCD address, which is the full path to the object through the PCD's tree structure. For example, the following is a PCD address:

```
pcd:portal_content/myFolder/stocks
```

The above address points to a PCD object called *stocks*, which is in a folder called *myFolder*, which is in the top-level folder *portal_content*.

In this example, the name *stocks* is the object's atomic name, which must be unique within the folder that contains the object (*myFolder*).

3.1.1.2 How to Manage Semantic Objects

This section describes the following tasks for managing all semantic objects:

- [Creating Objects \[Page 33\]](#)
- [Looking Up Objects \[Page 33\]](#)
- [Getting/Setting Attributes \[Page 33\]](#)
- [Deleting Objects \[Page 33\]](#)



Some updates to a semantic object require that you call `save()` on the affected semantic object (such as when updating attributes), while other changes do not require a call to `save()` (such as when adding iViews to a page). Check the Javadocs to determine if a call to `save()` is required.

3.1.1.2.1 Creating Objects

This section describes how to create a semantic object in the PCD by first creating a descriptor for the new object and binding it to a folder context.

A new object can be based on an existing PCD object (either as a copy or a delta link) or on an application already deployed to the portal.

Procedure

1. Create an instance of the helper object for the type of semantic object that you want to create, such as, `IiViews` for an iView, `ISystems` for a system, and so forth.

```
IiViews iViewSrv = (IiViews)
PortalRuntime.getRuntimeResources().getService(IiViews.KEY);
```

2. Create a descriptor for the new object that you want to create.

```
INewObjectDescriptor IVtoCreate = (INewObjectDescriptor)
iViewSrv.instantiateDescriptor(CreateMethod.NEW,
    "par:/applications/myProject/components/myComponent",
    request.getUser());
```

`instantiateDescriptor()` takes the following parameters:

Creating and Managing Content

- **New or Delta Link:** Indicates whether the new object is a copy of or a delta link to an existing object. Use constants from the `CreateMethod` class.
- **PCD Object or Application:** Indicates the existing PCD object or portal application on which to base the new object. The format can be one of the following:
 - **PCD address**, such as `pcd:portal_content/myFolder/myObject`.
 - **Application address**, in the following format:
`par:/applications/myApp/components/myComp`
 where *myApp* is the name of an application and *myComp* is the name of a component in myApp.

If you specify a PCD address, the first parameter can be either `NEW` or `DELTA_LINK`. If you specify an application, the first parameter must be `NEW`.
- **Current User**, specified by calling `getUser()` on the portal request object.

3. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

4. Perform a lookup of the folder in which you want to create the new object.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);

    String folderName = "pcd:portal_content/myFolder";
    Context ctx = (Context)iCtx.lookup(folderName);

    ...
}
```

5. Create the object by binding the descriptor for the new object to the folder context.

```
...

ctx.bind("myNewHelloIV", IVtoCreate);

}
catch (Exception e)
{
}
```

3.1.1.2.2 Looking Up Objects

This section describes how to look up a PCD object by performing a JNDI lookup and supplying the PCD address of the object.

Procedure

Creating and Managing Content

1. Create an instance of the helper object for the type of semantic object that you want to create, such as, `IiViews` for an `iView`, `ISystems` for a system, and so forth.

```
IiViews iViewSrv = (IiViews)
    PortalRuntime.getRuntimeResources().getService(IiViews.KEY);
```

2. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

3. Perform the lookup by supplying the PCD address of the object, and then cast the returned object to the appropriate semantic object interface.

```
InitialContext iCtx = null;
try
{
    String iViewID = "pcd:portal_content/myFolder/stocks";

    iCtx = new InitialContext(env);
    IiView result = (IiView) iCtx.lookup(objectAddress);
}
catch (Exception e)
{
}
```

3.1.1.2.3 Getting/Setting Attributes

This section describes how to get and set object attributes.

For more information on retrieving an object, see [Looking Up Objects \[Page 33\]](#).

Procedure

- Getting Attributes and Meta-Attributes

```
Response.write (obj.getAttribute ("attribute") );

Response.write (obj.getMetaAttribute ("attribute", "meta-attribute")
);
```

Text attributes generally require an additional parameter that indicates the locale, which can be obtained from the portal request object.

The following attributes require the use of the locale:

- **com.sap.portal.pcm.Title**
- **com.sap.portal.pcm.Description**

The following meta-attributes require the use of the locale:

- **plainDescription**
- **longDescription**
- **category**
- **validValueTitle0, validValueTitle1, and so forth.**

Creating and Managing Content

```
Response.write (obj.getAttribute ("attribute", request.getLocale() )
);
```

- Setting Attributes

```
obj.putAttribute("attribute", "value");
obj.save();
```

Attribute Constants

To specify an attribute, use the designated constant for that attribute. The constants for each semantic type are located in a corresponding interface in the `com.sap.portal.pcm.attributes` package.

For example, the constants for iView attributes are located in the `IAttriView` interface. The following code checks whether the current iView allows browser caching:

```
response.write(
    result.getAttribute(IAttriView.ATTRIBUTE_ALLOW_BROWSER));
```

3.1.1.2.4 Deleting Objects

This section describes how to delete a semantic object from the PCD.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the folder that contains the object that you want to delete.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);

    String folderName = "pcd:portal content/myFolder";
    Context ctx = (Context) iCtx.lookup(folderName);

    ...
}
```

3. Delete the object by unbinding the object from the folder that contains it. Use the atomic name of the object.

```
...

String atomicName = "myObject";
ctx.unbind(atomicName);

}
catch (Exception e)
{
}
```

}

3.1.1.3 How to Manage iViews

This section describes the following tasks for managing iViews:

- [Adding Related Items \[Page 33\]](#)

For information on creating, modifying and deleting iViews and other semantic objects, see [Managing Semantic Objects \[Page 33\]](#).

3.1.1.3.1 Adding Related Items

You can associate iViews with other iViews or pages, so that when an iView is displayed, links to its related iViews and pages are displayed in the side navigation panel.

The following types of related items are available:

- **Related Links:** Links to the related iViews and pages are displayed in the Related Links iView of the navigation panel.
- **Dynamic Navigation:** The content of the related iViews and pages are displayed in the Dynamic Navigation iView of the navigation panel.
- **Target Components:** Links to the related iViews and pages are displayed in the Drag&Relate Targets iView of the navigation panel.

For more information on Drag&Relate targets, see the SAP NetWeaver documentation on the Help Portal (<http://help.sap.com>) → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *Content Administration* → *Navigation* → *Drag&Relate Targets*.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the iView to which you want to add related items, and cast the object as an `IiView` object.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IiView myIiView = (IiView) iCtx.lookup(iViewId);
}
catch (Exception e)
{
}
```

3. Create a descriptor (`INewObjectDescriptor` object) for the iView or page that you want to add as a related link.

Creating and Managing Content

```
INewObjectDescriptor iViewDescriptor =
    (INewObjectDescriptor) iViewSrv.instantiateDescriptor
        (CreateMethod.DELTA_LINK,
         "pcd:portal_content/testxml", request.getUser());
```

4. Add the related item descriptor to the iView.

```
myIView.addRelatedItem(iViewDescriptor, "testxml",
    RelatedItemType.DYNAMIC_NAVIGATION);
```

3.1.1.4 How to Manage Pages

This section describes the following tasks for managing systems:

- [Adding Layouts to a Page \[Page 33\]](#)
- [Setting the Default Layout for a Page \[Page 33\]](#)
- [Adding iViews to a Page \[Page 33\]](#)
- [Removing iViews from a Page \[Page 33\]](#)

For information on creating, modifying and deleting iViews and other semantic objects, see [Managing Semantic Objects \[Page 33\]](#).

3.1.1.4.1 Adding Layouts to a Page

Each page is assigned a default layout, which is used for rendering the page. You can assign other layouts to a page in order to enable a user to personalize the page. Each user can select one of the other layouts assigned to the page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page to which you want to add an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage = (IPage) iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
}
catch (Exception e)
{
}
```

3. Create a descriptor (INewObjectDescriptor object) for the layout that you want to add to your page.

Creating and Managing Content

```
INewObjectDescriptor layoutToAdd =
    (INewObjectDescriptor) iViewSrv.instantiateDescriptor
        (CreateMethod.DELTA_LINK,
         "pcd:portal_content/templates/layouts/narrowWide",
         request.getUser());
```

4. Add the layout descriptor to the page.

```
myPage.addLayout(layoutToAdd, "newLayout");
```



If a layout with the same atomic name exists in the page, an error is thrown.

3.1.1.4.2 Setting the Default Layout for a Page

This section describes how to set a page's default layout. A page's default layout is the layout used for rendering if the page's layout has not been personalized.

Procedure

1. Set the parameters for the JNDI lookup.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page for which you want to set the default layout.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage = (IPage) iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
}
catch (Exception e)
{
}
```

3. Set the default layout by specifying the layout's atomic name.

```
myPage.setActiveLayout("narrowWideNarrow");
```

4. Save the changes.

```
myPage.save();
```

3.1.1.4.3 Adding iViews to a Page

This section describes how to add an iView (or page) to a page.

Procedure

Creating and Managing Content

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page to which you want to add an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage = (IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
}
catch(Exception e)
{
}
```

3. Create a descriptor (INewObjectDescriptor object) for the iView or page that you want to add to your page.

```
INewObjectDescriptor iViewDescriptor =
    (INewObjectDescriptor)iViewSrv.instantiateDescriptor
    (CreateMethod.DELTA_LINK,
        "pcd:portal_content/testxml", request.getUser());
```

4. Add the iView descriptor to the page.

```
myPage.addiView(iViewDescriptor, "testxml");
```

The iView is automatically displayed at the bottom of the left-most column of the layout that is currently being used for the page.

If you want to place the iView into a particular column of a particular layout, you can specify a layout container into which to add the iView. The following adds the iView into the `com.sap.portal.reserved.layout.Cont2` container (second column) of the current layout:

```
myPage.addiView(IVtoAdd, "testxml",
    "com.sap.portal.reserved.layout.Cont2");
```

`com.sap.portal.reserved.layout.Cont2` is the container ID of the second column as defined in the standard `narrowWideNarrow` layout.



If an iView with the same atomic name exists in the page, an error is thrown.

3.1.1.4.4 Removing iViews from a Page

This section describes how to remove an iView from a page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();
```

Creating and Managing Content

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page from which you want to remove an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage = (IPage) iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
    ...
}
```

3. Remove the iView from the page. Use the atomic name of the iView.

```
...

myPage.removeiView("myIView");
}
catch (Exception e)
{
}
```

3.1.1.5 How to Manage Layouts

The `ILayout` API enables you to get information about a layout and its containers:

- `getContainerID()`: Returns the ID for a specific container in the layout. Specify the container by its name as defined in the `portalapp.xml`.
- `getContaineriViews()`: Returns an array of strings that represent the atomic names of the iViews and pages within a specific container. Specify the container by its ID.
- `getContainerIDs()`: Returns an array of strings that represents the IDs of the containers in the layout.

The `ILayout` API also enables you to specify where within a container to place an iView that has already been added to the page, as described in [Adding iViews to a Page \(via Layout\)](#) [Page 33].

For information on creating, modifying and deleting iViews and other semantic objects, see [Managing Semantic Objects](#) [Page 33].

3.1.1.5.1 Adding iViews to a Page (via Layout)

This section describes how to position an iView within a container of a layout on a page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();
```

Creating and Managing Content

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
        IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page on which you want to position an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage = (IPage) iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
}
catch (Exception e)
{
}
```

3. Get a reference to the layout in which you want to move the iView, for example, by getting the default layout.

```
ILayout myLayout = myPage.getActiveLayoutObject();
```

You can also get a list of a page's layouts by calling `getLayouts()` on the page.

4. Position the iView by calling `setiViewContainer()` and specifying the iView that you want to move, a container, and the position within the container to which you want to move the iView. The first position is 0, the second is 1, and so forth.

```
myLayout.setiViewInContainer("testxml",
    "com.sap.portal.reserved.layout.Cont1", 1);
```

The above moves the `testxml` iView in the page to the second position in the `com.sap.portal.reserved.layout.Cont1` container of the default layout.

5. Save the changes to the layout.

```
myLayout.save();
```

3.1.1.6 How to Manage Systems

In the PCD, the portal stores systems, which are sets of properties that represent an external back-end application. Systems, such as for JDBC-compliant databases or SAP R/3 systems, enable connections to specific applications of these types and the retrieval of data.

For more information on systems, see the SAP NetWeaver documentation on the Help Portal (<http://help.sap.com>) → SAP NetWeaver → People Integration → Portal → Administration Guide → System Administration → System Configuration → System Landscape.

This section describes the following tasks for managing systems:

- [Getting/Setting System Aliases \[Page 33\]](#)
- [Getting User Mapping \[Page 33\]](#)
- [Getting Aliases for All Systems \[Page 33\]](#)

For information on creating, modifying and deleting iViews and other semantic objects, see [Managing Semantic Objects \[Page 33\]](#).

3.1.1.6.1 Getting/Setting System Aliases

This section describes how to get and set system aliases.

- Get a System's Aliases

```
String[] aliases = result.getAliases();
```

- Add/Remove an Alias for a System:

```
result.addAlias("alias2");
```

```
result.removeAlias("alias1");
```

- Set a System's Default Alias:

```
result.changeDefaultAlias ("alias2");
```

3.1.1.6.2 Getting User Mapping

This section describes how to retrieve the user mapping (that is, the user name and password) associated with a system for the current user. The portal tries to connect to the back-end system using this user name and password when the current user requests an iView that connects with the system.

Procedure

1. Get the `ISystemUserData` object associated with the system for the current user.

```
ISystemUserData userMappingData = result.getUserMappingData(  
    request.getUser());
```

2. Get the user name or password from the `ISystemUserData` object.

```
userMappingData.getUser();  
userMappingData.getPassword();
```

3.1.1.6.3 Getting Aliases for All Systems

The `ISystems` interface provides the following methods for retrieving aliases for all systems in the PCD (in the code samples, `systems` is an object whose class implements the `ISystems` interface):

- All Aliases:

```
String[] aliases = systems.getAliases();
```

- All Default Aliases:

```
String[] defaultAliases = systems.getDefaultAliases();
```

The following retrieves the `ISystems` service:

```
ISystems systemSrv = (ISystems)PortalRuntime.getRuntimeResources()  
    .getService(ISystems.KEY);
```

3.1.1.7 Essential Information

This section provides the following information to help you manage systems:

Creating and Managing Content

- [portalapp.xml \[Page 33\]](#): Information about the `portalapp.xml` file for an application that works with systems.
- [JARs and Packages \[Page 33\]](#): The packages and JAR files required for managing systems.

3.1.1.7.1 portalapp.xml

In order to use the semantic object APIs, you must create the following service references in the `portalapp.xml` file:

```
<application-config>
  <property name="ServicesReference" value="com.sap.portal.ivs.api_iview,
com.sap.portal.ivs.api_landscape" />
</application-config>
```

3.1.1.7.2 JARs and Packages

This section lists the packages used in this project and the JAR files required for compilation.

JARs (for compilation)

- `com.sap.portal.ivs.api_iview_api.jar`
- `com.sap.portal.ivs.api_landscape_api.jar`

Packages

- `com.sap.portal.pcm.iView`
- `com.sap.portal.pcm.page`
- `com.sap.portal.pcm.layout`
- `com.sap.portal.pcm.system`
- `com.sap.portal.pcm.attributes`

Other Classes

The following is a list of additional classes used in this section and the corresponding packages:

Class	Package
Constants	<code>com.sap.portal.directory</code>
Context	<code>javax.naming.Context</code>
InitialContext	<code>com.sapportals.portal.prt.jndisupport</code>
IPcdContext	<code>com.sapportals.portal.pcd.gl</code>
PcmConstants	<code>com.sap.portal.pcm.admin</code>

3.1.2 Working with XML

The portal provides several ways of working with XML content and performing transformations:

- **XML iViews:** The portal comes with an XML iView template, that enables administrators to specify an XML source and one of the built-in transformers. The iView displays the transformed content.

This method is for administrators who are working with an HTTP-based XML source, whose XML content can use one of the built-in transformers.

For more information, see [Creating XML iViews \[External\]](#).

- **Creating Transformers:** You can build additional transformers for the XML iView. The custom transformers are displayed to administrators in the XML iView wizard.

For more information creating transformers, see [Providing Transformers \[Page 33\]](#).

- **Using the Transformation Service:** You can create a portal component that performs its own transformations, with the help of the transformation service.

For more information on using the transformation service, see [Transformation Service \[Page 33\]](#).

- **Using the Content Provider Framework:** The portal provides a framework for displaying third-party XML content.

For more information, see [Displaying External XML-Based Content \[Page 33\]](#).

3.1.2.1 Transformation Service

Purpose

The transformation service used to transform data in XML format with existing transformers/style sheets.

The transformation service is based on the SAP XML toolkit that implements JAXP (see <http://java.sun.com/xml/jaxp/index.jsp> for more details). It supports all JAXP XML source and result formats and provides the `HTTPStreamSource` source. Based on the content fetching service, the `HTTPStreamSource` source allows caching, proxy settings and access to properties defined in the Portal Content Directory (PCD). The transformation service comes with built-in transformers.

Transformation service interface: `ITransformerService`

Transformation service key: `ITransformerService.KEY`

The transformation service requires basic skills in the Extensible Stylesheet Language (XSL).

3.1.2.1.1 XML Transformation

XML transformation is done with XSL and SAX transformers. With the transformation service you define the transformers once and then use them in combination with already defined transformers without needing to re-define them again before transforming. Transformation can be done on registered transformers only. Registered transformers can also be used by other content developers using the XML iView user interface from the portal.

Combining several transformers to generate one output is called `Pipe Transformation`. The `Pipe transformation` allows you to develop light weight transformers and connect them to one powerful transformer.

Creating and Managing Content

A transformer has the following attributes collected by interface
ITransformerInformation:

Attribute	Type	Description
Name	String (cs)	The name of the transformer. The name has to be unique for an application component.
Component name	String (cs)	The name of the portal component or portal service that registered the transformer.
Version	Float	The version of the transformer. Using the correct version number will avoid errors when a transformer is upgraded.
Transformer type	TransformerType	Registration type of the transformer. There are the following types: <ul style="list-style-type: none"> • Built-in Transformer was provided by the transformation service. • Persistent Transformer was supplied by transformation provider (see chapter Providing Transformers [Page 33]). • Temporary Transformer is temporally supplied from regular portal application.
From scheme	String (cs)	XML input format. The scheme from which the transformer gets the XML data..
To scheme	String (cs)	XML output format. Defines the output format of the transformer.
Description	String	General description of the transformer.

Activation

Following steps are necessary to activate a XML transformation:

1. Create a Transformer List
To get related transformers information according to the attributes, use the requested attributes as parameter of the method `getTransformersInformation()`. You should call this method for each transformer you want to add. The method returns all transformers that match the specified attribute.
2. Defining parameters
Create an array with the same size of the transformer list. Every item in the array must be a map of parameters related to the transformer with the same index.
3. Generate the XML source
Generate the XML source to be transformed. See *XML Source* for more details.
4. Generate XML result
Generate the object that will receive the result of the transformation. See *XML Result* for more details.
5. Activate transformation

Call method `transform()` of the service.

XML Source

The implementation of XML source must inherit from `javax.xml.transform.Source`. The XML input can be file, stream, DOM or another customized class. The source classed provided by the JDK are:

- `javax.xml.transform.stream.StreamSource`
- `javax.xml.transform.sax.SAXSource`
- `javax.xml.transform.dom.DOMSource`

See the JDK Documentation for more details.

The `HTTPStreamSource` class, an addition of the transformation service, is used for working with URL based XML sources that uses the portal cache, proxy settings and PCD attributes to load XML files from the web. It is strongly recommended to work with this class when loading XML files from the web. For more details see the class definition in the transformation service Javadoc and examples.

XML Result

The implementation of XML result must inherit from `javax.xml.transform.result`. The class supplied keeps the result of the transformation in the format specified by the last transformer in the transformation pipe. Result classes provided by the JDK are:

- `javax.xml.transform.stream.StreamResult`
- `javax.xml.transform.sax.SAXResult`
- `javax.xml.transform.dom.DOMResult`

See the JDK Documentation for more details.

Result Types

There are the following XML results:

- `IPortalComponentResponse`

in-order for the results to be written directly to the response of the portal component you should create a new `StreamResult` object with the response writer.

Example:

```
...
// Setting result stream
StreamResult strmResult = new StreamResult(response.getWriter());
// Transforming
tService.transform(src,trns, paramsArr, context, null, strmRes);
...
```

- `HTMLB`

Creating and Managing Content

When working with the XHTMLB transformer the result of the transformation is a set of HTMLB objects. Default behaviour is to create this set of objects in the page context, however it is possible to specify that the result should be rendered to the response by parameters (see [Built-in Transformers \[Page 33\]](#) for more details).

- String

In order for the result to be written to a string you should create a `StreamResult` object with a `ByteArrayOutputStream` as its constructor input field. After the transformation is finished get the string from the created `ByteArrayOutputStream` object.

Example:

```
OutputStream outStrm = new ByteArrayOutputStream();
StreamResult strm = new StreamResult(outStrm);
tService.transform(source, trns, params, context, null, strm);
.
.
tService.transform(source, trns, params, context, null, strm);
.
.
String result = outStrm.toString();
```

3.1.2.1.2 Providing Transformers

A transformer provider has a portal archive (PAR) file that contains the transformers and is responsible for registering and removing the transformers from the list of transformers in the transformation service. All transformers will be registered with type `persistent`. The provider can contain several XSL and SAX transformers.

Default implementation class of the transformer provider PAR:

```
com.sap.portal.httpconnectivity.transformationsservice.TransformersProvider
```

All customized classes inherit from this class.

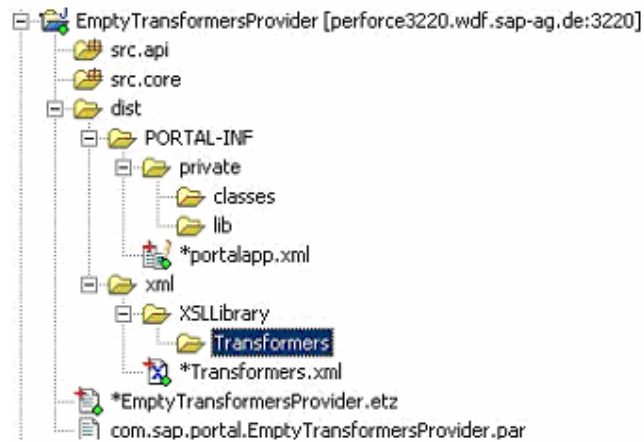
3.1.2.1.2.1 Simple Providers

For a simple provider you do not have to implement Java code. Simple providers can only contain XSL transformers. Simple transformers cannot use language properties (`ResourceBundle` classes) and cannot customize the default behavior.

PAR Structure

The PAR file must have the following folder structure:

Creating and Managing Content



The folder named *Transformers* contains all the XSL files. Note that there are no Java files in the project.



If you want to use different folders you have to adjust the *portalapp.xml* file accordingly.

Portalapp.xml File

The *portalapp.xml* registers the provider with the PRT registry mechanism, declares the provider service and general application configurations.

- Portal registry definition

Defines this PAR as a provider in the registry entry. The *path* attribute must be set as follows:

```
<registry>
  <entry
    path="runtime/transformers/com.sap.portal.EmptyTransformersProvider"
    name="TransformersProvider"
    type="service"/>
</registry>
```

- Service configuration

Declare the provider as a service. The service must be defined as follows:

Creating and Managing Content

```
<services>
  <service
    name="TransformersProvider">
    <service-config>
      <property
        name="className"
        value="com.sap.portal.httpconnectivity.transformation.service.
          TransformersProvider"/>
      <property
        name="classNameFactory"
        value=""/>
      <property
        name="classNameManager"
        value=""/>
      <property
        name="SecurityZone"
        value="com.sap.portal/no_safety"
      />
    </service-config>
  </service>
</services>
```

- Application configuration

Set the `startup` property to `true` so that the PRT will register the provider when deploying and not the first time that the service is called.

Example:

```
<application-config>
  <property
    name="ServicesReference"
    value="com.sap.portal.htmlb,com.sap.portal.transformation.service"/>
  <property
    name="releasable"
    value="false" />
  <property
    name="startup"
    value="true" />
</application-config>
```

- Transformers.xml File

This file contains the provided transformers. The file must be located at `<par folder>/dist/xml/transformers.xml`. It is divided into a XSL and SAX part. Every transformer has a name, a description, from/to scheme, source name and version (See [XML Transformation \[Page 33\]](#) for more).

For an XSL transformer the source name property is the XSL file and in case of an SAX transformer it is the class name.

Example:

Creating and Managing Content

```
<?xml version="1.0" encoding="utf-8"?>
<transformation-resources>
  <transformers
    type="XSL">
      <!-- Holder of XSL transformers -->
      <transformer>
        <!-- Represet single transformmr -->
        <property
          name="Name"
          value="MY_RSS_TO_XHTMLB" />
        <!-- Key of the transformer-->
        <property
          name="Description"
          value="Transform RSS files to XHTMLB" />
        <property
          name="FromURI"
          value="RSS" />
        <!-- Source scheme URI -->
        <property
          name="ToURI"
          value="XHTMLB" />
        <!-- result scheme URI -->
        <property
          name="SourceName"
          value="RSS_TO_XHTMLB.xsl" />
        <!-- XSL file name -->
        <property
          name="Version"
          value="1.0" />
      </transformer>
    </transformers>
  <transformers
    type="SAX">
      <!-- Holder of the SAX transformers -->
      <transformer>
        <property
          name="Name"
          value="MY_UNIQUE_ID_ADDER" />
        <property
          name="Description"
          value="Add an unique id for every node in the XML" />
        <property
          name="FromURI"
          value="XML"/>
        <property
          name="ToURI"
          value="RSS"/>
        <property
          name="SourceName"
          value="com.sapportals.portal.httpconnectivity.
            saxtransformerprovider.MyUIDAdderHandler" />
        <!-- Full class name -->
        <property
          name="Version"
          value="1.0" />
      </transformer>
    </transformers>
  </transformation-resources>
```

3.1.2.1.2.2 SAX Providers

Steps to register a SAX transformer in the transformation service.

Implementation

Content developers must create their own class derived from `TransformersProvider` class and overwrite the method `getSAXHandler(ITransformerInformation info)` that returns a new instance of SAX handler inherited from `EPSAXDefaultHandler`. A simple implementation of the method is to take the class name from the `ITransformerInformation` and instantiate it.

Portalapp.xml File

The only change from the settings for a [simple provider \[Page 33\]](#) is the class name of the service.

3.1.2.1.2.3 Resource Bundle Holders

The transformation service inserts a resource bundle object with the right language to every transformer as a parameter before transforming. The parameter name of the transformer is `ResourceBundle`. When no resource bundle is defined the default transformation service resource bundle will be used.

Implementation

Content developers must create their own class inherited from the `TransformersProvider` class and overwrite the method `getResourceBundle(Locale locale)`. This method will be called at run-time before the transformation process is started. The result class must be `PropertyResourceBundle`. The overwritten method must be in the core section of the PAR. The creation of the localization files/resource bundles is the same as for other PARs using localization

Portalapp.xml file

The only changes from the settings for a [simple provider \[Page 33\]](#) is the class name of the service and the property `ResourceBundleName` in the `service-config` section.

3.1.2.1.2.4 Customizing Options

All the customization options require implementation of a class that extends the `TransformersProvider` class and change the class name property in the `portalapp.xml` file.

Changing Folder Structure

To place the `Transformers.xml` in a different location overwrite the method `getTransformersResourcePath()`. This method returns a full path name to a file holding the list of transformers.

Creating and Managing Content

To change the location of the XSL files overwrite the method `getXSLTransformerPath(String tXSLPath)`. This method receives the `SourceName` property from the `Transformers.xml` file and returns the full path name to the XSL file.

Adding Transformers by Coding

Transformer providers can add transformers without defining them in the `Transformers.xml` file by overwriting the `init(IServiceContext serviceContext)` method and implement in one of the following ways:

1. Call the default implementation, `super(serviceContext)`. Create the new transformer information using the `createTransformerInformation()` method of the transformation service. Set the transformer by calling the appropriate `setTransformer()` method in the service.
2. Load transformers information by calling `loadTransformers()` method and hold the received list. Create your transformer using `createTransformerInformation()` of the service and add the `ITransformerInformation` result to the list of transformers. Call method `setTransformers()` with the list of transformers information.

This customization is useful when there is additional information needed to add a transformer or when your provider is also servicing other purposes.

Add Packages to the SAX Class Names

Overwrite method `String getSAXClassName(String tClassName)` that receives the value of the `SourcePath` property in the transformer definition in the `transformers.xml` file. This method must return a full qualified class name including the package name.

Standalone Provider

This is a provider that does not inherit from class `com.sap.portal.httpconnectivity.transformationsservice.TransformersProvider`. When you develop a standalone provider you must supply the following functionalities:

1. Provider must be a portal service.
2. The provider must be registered when deployed in the `PortalRegistry`.
3. Registration of the transformers when initialized. Transformers must be added using the `ITransformersService` methods.
4. Un-Register a transformers when a provider is removed or updated.
5. Un-Register from the `PortalRegistry`.

3.1.2.1.2.5 Creating a Provider Step-by-Step

Following steps are necessary to create a PAR file for a transformer provider:

- Create a simple empty provider PAR as defined above or use an example from the PDK.
- Edit `portalapp.xml` according to the steps previously described ([Simple provider \[Page 33\]](#), [SAX Provider \[Page 33\]](#)).
- Create/Edit the `transformers.xml` file.

Creating and Managing Content

- Create the XSL and SAX transformers and add them to the PAR file.
- If needed, implement SAX loader.
- Implement the `ResourceBundle` provider and create the localization files (if needed).
- Implement a customized implementation (if needed).
- Deploy the PAR file.
- Now your transformers must be available in the XML iView wizard and editor.

3.1.2.1.2.6 Declaring Transformer Parameters

When adding transformers, the provider can define parameters that will control the behavior of the transformer. These parameters are visible in the XML iView editor. The rules that should be applied when providing parameters in transformer are as follows:

Parameters types

Providers can decide that certain parameters should not be exposed to the editor by not putting the parameters in the list of parameters.

SAX Transformers Parameters

Every SAX handler must implement the `ITransformerProperties` interface that declare the method `getInputProperties()`. This method returns a map with the editable parameters of the SAX handler. The map keys are the names of the parameters. The values of the map can be of two types:

- String objects representing the default value.
- List of objects representing the valid options of the parameter. The first value in the list is the default value.

XSL Transformers Parameters

The parameters are declared in the standard XSL way, using `xsl:param` elements. Controlling the type of the parameter is done by using two additional attributes:

- Type `sap:param`

This entry controls if this parameter is editable. Possible values are:

- `hidden`
Non-editable parameter
- `visible` (default)
Editable parameter.

Example:

```
<xsl:param name="ResourceBundle" sap:param-
type="hidden"/>
```

- Option `sap:param-options`

This entry declares a list of possible values for the parameter.

Syntax: `sap:param-options="<default value>;<second value>;..."`.

Example:

```
<xsl:param
    name="ShowDatesMode"
    sap:param-
    options="MAIN_ONLY;ALL;ITEMS_ONLY">ITEMS_ONLY
</xsl:param>
```



When using XSL parameters you must declare the namespace accordingly.

Example:

```
xmlns:sap="http://www.sap.com/2004/Transformers/1.0".
```

3.1.2.1.3 Built-in Transformers

The transformation service provides built-in transformers that give default solution to the following formats:

- XHTMLB – see specification below
- RSS – see <http://web.resource.org/rss/1.0/spec> for more details
- Busdoc XML.

All built-in transformers component name is

`ITransformerService.BUILT_IN_TRANSFORMERS_KEY` and the transformer type is `BUILT-IN`.

3.1.2.1.3.1 XHTMLB Transformer

The XHTMLB transformer is a SAX transformer that transforms between XHTMLB (see the [XHTMLB Specification \[Page 33\]](#) for details about the format) and HTMLB DOM. The result of the transformation is a set of HTMLB objects ready to be rendered or to be included to other HTMLB objects. If you provide an HTMLB container it will contain the created HTMLB objects. If no container is provided, the transformer will create an HTMLB form document. The component Id is generated when the ID attribute for the XHTMLB element is not specified.



This transformer must be the last one in the transformer list.

The attributes of the XHTMLB transformer are as follows:

Attribute	Value
Name	XHTMLBSAXHandler
Version	1.0
From scheme	XHTMLB
To scheme	HTML

Parameters

The parameters have the format:

```
ITransformerService.BUILT_IN_TRANSFORMERS_KEY.<parameter>
```

Following parameters are available:

Parameter	Type (Default value)	Description
DebugMode	Boolean (<i>false</i>)	<p>In DebugMode (value set to true), debug logs and traces are appended at the end of the top container. The logs include error notifications, trace of every component that was created and general status information.</p> <p>An enabled DebugMode slows down the transformation process.</p>
HTMLBTopContainer	Inherited from com.sapportals.htmlb.Container	<p>Top container that contains all the HTMLB objects that are created in the transformation process of the XHTMLB document.</p> <p>When no value is specified, the transformer will try to get the HTMLB document from the page context. If no document can be found, it will create a form document as the top container. You can get the created document from the page context object (method <code>IPageContext.getDocument()</code>).</p>
HTMLBDeclaredComponents	java.util.Map (<i>null</i>)	<p>This parameter specified a Map object that receives the components with a given ID. The Map object can be used to get the objects and manipulate them.</p> <p>When no Map is specified, the declared components are treated as any other component.</p>
HTMLBRenderAtEnd	Boolean (<i>true</i>)	<p>This flag defines if the created document to the response is rendered when the transformation is finished.</p>

Error Handling

When the DebugMode is enabled, all error and trace messages are placed as plain HTML text at the end of the top container. No exception is thrown when a method or parameter is used that do not exist.

Not Supported HTMLB Components:

Following HTMLB components are not supported:

Creating and Managing Content

- MenuBar
- MenuItem
- Chart
- FormLayout
- DragSource
- DropTarget
- EventValidationComponent
- EventValidationContainer
- JavaScriptFragment
- PopupTrigger
- RoadMap
- RoadMapItem

3.1.2.1.3.1.1 XHTMLB Specification

XHTMLB is XML based format for creating HTMLB documents using the [XHTMLBTransformer \[Page 33\]](#). To creating XHTMLB documents you need knowledge of [HTMLB \[Page 33\]](#).

HTMLB Components Declaration

Every node in XHTMLB is equivalent to an HTMLB component. Every attribute of an XHTMLB element is equivalent to a call of a set method, where the name of the attribute is the name of the method and the attribute value is the method parameter.

The text in the XML node is added to the component by using the method `setText()` or `setValue()`. If the current component is an HTMLB container, the method `addText` will be called. The last text node is used as the value.

Example of an XHTMLB node:

The following statement will create an HTMLB button component with the text `My button` and with the value `MyEventValue` that will be sent when clicking on the button (OnClick event).

```
<Button id="testButton" OnClick="MyEventValue">My  
button</Button>
```

Example for multiple text nodes:

```
<TextView Wrapping="true">
  This text will be overwritten
  <parameter
    name="Design"
    value="LABEL"
    class="enum.TextViewDesign" />
  This text will appear in the TextView component
</TextView>
```

Parameters Declaration

Declaring parameters has the following reasons:

- Calling a method on the current component that has a static input field value.
- Adding key/value pair to the current component (when needed).

To call a method that requests a static value from a class (like enumerated values), add the element named `parameter` with the following attributes:

- Name
The name of the method without the “set” or “add” prefix.
- class
The name of the class.
- package
The name of the package (default package name is `com.sapportals.htmlb`)
- value
The name of the static member that will be inserted to the method.

Example for calling a method with a static field of a class as a value:

The following example generates an HTMLB TextView component. The method `setDesign()` will be called with `com.sapportals.htmlb.enumTextViewDesign.LABEL` class as parameter. The method `setLayout()` will be called with `com.sapportals.htmlb.enumTextViewLayout.BLOCK` class as parameter.

```
<TextView
  Wrapping="true">
  <parameter
    name="Design"
    value="LABEL"
    class="enum.TextViewDesign" />
  <parameter
    name="Layout"
    value="BLOCK"
    class="enum.TextViewLayout" />
  Some text with label design and block layout
</TextView>
```


Example for calling a method with a key/value parameter:

The key/value pair parameter is used for HTMLB components that have a selection list, like ListBox, DropDownListBox, ToolbarDropDownListBox and BreadCrumb.

```
<DropDownListBox id="DropDown1"
  width="120" text="DropDown">
  <parameter
    key="Item1"
    value="Value one"/>
  <parameter
    key="Item2"
    value="Value two"/>
</DropDownListBox>
```

HTML Tags

HTML tags will be added to the current HTMLB container. HTMLB components cannot be placed in HTML tags; once an HTML tag is defined all its children will be HTML elements as well.

Example:

In this example the HTML `span` tag and its children will be added to the HTMLB `Group` container.

```
<Group>
  <ButtonRow>
    <Button
      id="tst7"
      width="100">Test1</Button>
    <Button
      id="tst8"
      width="100">Test2</Button>
  </ButtonRow>
  <span>
    <a
      href="http://www.cnn.com">HTML Link</a>
  </span>
</Group>
```

Creating XHTMLB Tables

XHTMLB table are created by using a TableView element with a JCOTableViewModel or DefaultTableViewModel. The models can have TableColumn elements and TableRow elements. TableRow elements are no HTMLB components and can have TableCell elements which are also no HTMLB components.

Creating and Managing Content

Example:

```

<TableView ...>
  <JCOTableViewModel ...>
    <TableColumn index="0" name="FirstRow" ...>
    </TableColumn>
    <TableColumn index="1" name="SecondRow" ...>
    </TableColumn>
    <TableRow ...>
      <TableCell>Row one col one</TableCell>
      <TableCell>Row one col two</TableCell>
    </TableRow>
    <TableRow>
      <TableCell>Row two col ones</TableCell>
      <TableCell>Row two col two</TableCell>
    </TableRow>
    <TableRow>
    </TableRow>
    </TableRow>
    ...
  </JCOTableViewModel>
</TableView>

```

TableView Element

The TableView element can have all the `set` methods (XHTMLB attributes and parameter elements) like any other HTMLB element except the following:

- `setModel`

It is used to insert the TableViewModel by parameter. When the model attribute is set for a TableView element, the value must point to a parameter inserted to the XHTMLB transformer with an object inherited from TableViewModel.

- `CellRenderer`

XHTMLB supports self defined cell renderers. The cell render object must inherit from `com.sap.portal.transformationservice.xhtmlb.IXHTMLBCellRenderer`. If no cell renderer is defined, the data will be treated as raw data. The definition can be done by adding a parameter element to the TableView element with the class definition in the class and package attribute or with the name of the external property in the value attribute. Only one cell renderer can be defined for a table, the last one specified will be used. To use a self defined renderer the cell must have the attribute `useRenderer="true"`.

Example:

```

<TableView id="TestTable" width="400">
  <parameter name="CellRenderer"
    package="com.sap.portal.httpconnectivity.transformationservice"
    class="xhtmlb.DefaultXHTMLBCellRenderer" />
  ...
  <TableRow selectRow="true">
    <TableCell>http://www.cnn.com</TableCell>
    <TableCell useRenderer="true">
      <![CDATA[<div style="font-size:x-small">
        <a href="http://www.google.com">google</a>
        </div>]]>
    </TableCell>
  </TableRow>
</TableView>

```

- Row specific methods must be defined in the TableRow element.
- Cell specified methods must be defined in the TableCell element. The cell methods get the row index, cell index and a value to set.

JCOTableViewModeler/DefaultTableViewModel Elements

A model can have no attributes and parameters.

TableColumn Element

Supports all the `set` methods of the HTMLB TableColumn component. Every table column must have an index (starting from 0) and a name. A TableColumn element has no child elements.

TableRow Element

The TableRow element has no corresponding HTMLB component. It is used to wrap the content of a table row. It can receive all the methods related to rows. Row methods are:

- `selectRow`
- `setOnRowSelection`
- `setRowSelectable`
- `setRowVAlignment`.

TableCell Element

The TableCell element has no corresponding HTMLB component. It holds the actual data of the cell and has all “set” methods attributes of the TableView component that refers to cells. These methods take row index, column index and value to set. Cell methods are:

- `setCellDisabled`
- `setCellHAlignment`
- `setCellInvalid`
- `setCellType`
- `setCellVAlignment`
- `setColspanForCell`
- `setRowspanForCell`
- `setStyleForCell`.

Table Toolbar

The Table Toolbar element must be declared before the starting the declaration of the TableView. The connection is done by assigning the Id to the toolbar element and adding the attribute `toolbar="<toolbar element ID>"` to the TableView element.

TableView Limitations:

Following limitations apply to the TableView:

- Cells cannot have child elements. As a result, no HTMLB component can be a child of a table cell.
- All the table columns must be defined before the table rows.
- Sorting of columns is not supported by default.

Examples

[Example using a GridLayout component \[Page 33\]](#).

[Example using a TableView component \[Page 33\]](#).

XHTMLB Differences to HTMLB Components

Following components have a different behaviour:

- GridLayoutCell
All elements must have row and column attribute with an integer value starting at 1. See the [GridLayout example \[Page 33\]](#) for more details.
- Group
To define a group child element as header component, add the attribute `isHeaderComponent = "true"` to the child element.
- ItemList
Each component in the list can set its bullet image URL by adding the attribute `bulletURI="<bullet URL>"` and set the bullet style by adding the attribute `style="<style parameters>"`.
- TableView
TableView and its related components are defined completely different than in HTMLB. Refer to the [TableView example \[Page 33\]](#) for more details.
- TabStripItem
To set a TabStripItem child element as header component, add the attribute `isHeader="true"` to the child element. Other child elements will be inserted to the body of the TabStripItem by the order they are specified.

Not Supported HTMLB Components

Following HTMLB components are not supported:

- MenuBar
- MenuItem
- Chart
- FormLayout
- DragSource
- DropTarget
- EventValidationComponent
- EventValidationContainer
- JavaScriptFragment
- PopupTrigger
- RoadMap
- RoadMapItem

3.1.2.1.3.1.1.1 XHTMLB GridLayout Example

Example places different HTMLB components in a GridLayout:

Creating and Managing Content

```

<?xml version="1.0"?>
<GridLayout>
  <GridLayoutCell id="cell11" row="1" column="1" width="100" height="100">
    <ButtonRow>
      <Button id="tst1" width="100">Test1</Button>
      <Button id="tst2" width="100">Test2</Button>
    </ButtonRow>
  </GridLayoutCell>
  <GridLayoutCell id="cell12" row="1" column="2" width="100" height="100">
    <Tray>
      <ButtonRow>
        <Button id="tst3" width="100">Test4</Button>
        <Button id="tst4" width="100">Test5</Button>
      </ButtonRow>
      <DropDownListBox id="DropDown1" width="120" text="DropDown">
        <parameter key="Item1" value="Value one"/>
        <parameter key="Item2" value="Value two"/>
      </DropDownListBox>
    </Tray>
  </GridLayoutCell>
  <GridLayoutCell id="cell13" row="2" column="1" width="100" height="100">
    <ItemList>
      <Group tooltip="group one tooltip">
        <Button isHeaderComponent="true" id="tst6" width="100">Test5</Button>
        <a href="http://www.cnn.com">HTML Link</a>
      </Group>
      <Group>
        <ButtonRow>
          <Button id="tst7" width="100">Test1</Button>
          <Button id="tst8" width="100">Test2</Button>
        </ButtonRow>
        <a href="http://www.cnn.com">HTML Link</a>
      </Group>
    </ItemList>
  </GridLayoutCell>
  <GridLayoutCell id="cell13" row="3" column="1" width="100" height="100">
    <Tree id="Tree1">
      <TreeNode id="Node1" Text="Node 1 text">
        <TreeNode id="Node3" Text="Node 2 text">
          <TreeNode id="Node2" Text="Node 3 text">
            <Button id="tst9" width="100">Test2</Button>
          </TreeNode>
        </TreeNode>
      <TreeNode id="Node4" Text="Node 4 text"/>
    </TreeNode>
  </Tree>
  </GridLayoutCell>
  <GridLayoutCell id="cell14" row="3" column="2" Width="100%" height="100">
    <FlowLayout>
      <CheckboxGroup id="CheckboxGroup1" ColumnCount="6">
        <Checkbox id="Checkbox1" value="Checkbox one"/>
        <Checkbox id="Checkbox2" value="Checkbox two"/>
        <Checkbox id="Checkbox2" value="Checkbox three"/>
      </CheckboxGroup>
      <TextView Labeled="true" Text="Test Text"/>
    </FlowLayout>
  </GridLayoutCell>
  <GridLayoutCell id="cell15" row="4" column="1" width="100" height="100">
    <FlowLayout>
      <RadioButtonGroup id="RadioGroup1" key="TestRadio" ColumnCount="10">
        <RadioButton id="Radio1" key="TestRadio0" selected="true">
          <InputField sid="Input0" value="test input2"/>
        </RadioButton>
        <RadioButton id="Radio2" key="TestRadio1"
          text="Radio two" Labelled="true">
        </RadioButton>
      </RadioButtonGroup>
    </FlowLayout>
  </GridLayoutCell>

```

Result

The image displays a complex web form layout. At the top left, there are two buttons labeled 'Test1' and 'Test2'. Below them is a 'Test5' button, which is part of a dropdown menu currently showing 'Value one'. To the right of this is another section with three unchecked checkboxes labeled 'Checkbox one', 'Checkbox two', and 'Checkbox three', followed by a 'Test Text' label. Below the checkboxes is a large text area containing the text 'aaaaa > bbbbb > ccccc > ddddd > eeeee > ffffff'. On the left side of the form, there is a sidebar with a tree view structure showing 'Node 1 text', 'Node 2 text', and 'Node 4 text'. At the bottom of the form, there is a section with a radio button labeled 'test input2', a 'Radio two' label, and three text input fields labeled 'Text 2', 'Text 3', and 'test input'.

3.1.2.1.3.1.1.2 XHTMLB TableView Example

Example using a TableView and a JCOTableView model:

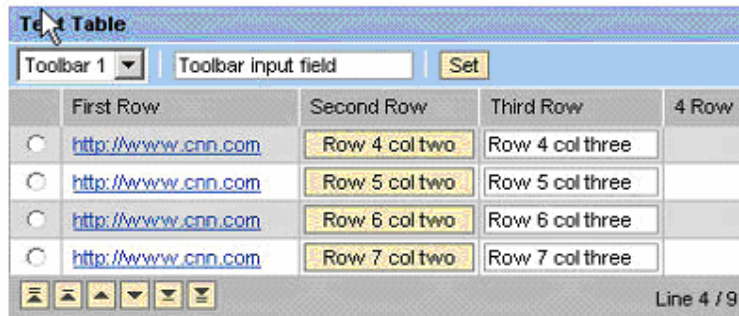
Creating and Managing Content

```

<?xml version="1.0"?>
<FlowLayout>
  <Toolbar id="TestTableToolbar">
    <ToolbarDropDownListBox id="tbrLstBox">
      <parameter key="TestVal1" value="Toolbar 1"/>
      <parameter key="TestVal2" value="Toolbar 2"/>
    </ToolbarDropDownListBox>
    <ToolbarSeparator/>
    <ToolbarInputField id="tbrInpFld" value="Toolbar input field"/>
    <ToolbarSeparator/>
    <ToolbarButton id="tbrBttn" text="Set" value="SomeValue"/>
  </Toolbar>
  <TableView id="TestTable" width="400"
    Summary="Table summary" footerVisible="true"
    VisibleFirstRow="5" VisibleRowCount="4"
    OnNavigate="NavigateTestTable" HeaderText="Test Table"
    OnHeaderClick="DoClicked" toolbar="TestTableToolbar">
    <parameter name="CellRenderer"
      package="com.sap.portal.httpconnectivity.transformationservice"
      class="xhtmlb.XHTMLBCellRenderer"/>
    <parameter name="NavigationMode" value="BYLINE"
      class="enum.TableNavigationMode"/>
    <parameter name="Design" value="ALTERNATING"
      class="enum.TableViewDesign"/>
    <parameter name="SelectionMode" value="SINGLESELECT"
      class="enum.TableSelectionMode"/>
  </JCOTableViewModel>
  <TableColumn index="0" name="FirstRow" title="First Row"
    LinkClickTarget="_blank" LinkColumnKey="FirstRow"
    TooltipForColumnHeader="First row tooltip">
    <parameter name="Type" value="LINK" class="enum.TableColumnType"/>
  </TableColumn>
  <TableColumn index="1" name="SecondRow" title="Second Row"
    CellsDragable="true">
    <parameter name="Type" value="BUTTON" class="enum.TableColumnType"/>
  </TableColumn>
  <TableColumn index="2" name="ThirdRow" title="Third Row">
    <parameter name="Type" value="INPUT" class="enum.TableColumnType"/>
    <parameter name="DropTargetDesign" value="BORDERED"
      class="enum.DropTargetDesign"/>
  </TableColumn>
  <TableColumn index="3" name="4Row" title="4 Row" OnDrop="DropOnRow4"
    Width="400" LinkClickTarget="_blank" LinkColumnKey="4Row">
    <parameter name="Type" value="TEXT" class="enum.TableColumnType"/>
    <parameter name="DropTargetDesign" value="UNDERLINED"
      class="enum.DropTargetDesign"/>
  </TableColumn>
  <TableRow>
    <TableCell>Row one col one</TableCell>
    <TableCell>Row one col two</TableCell>
    <TableCell>Row one col three</TableCell>
    <TableCell>Row one col three</TableCell>
  </TableRow>
  <TableRow>
    <TableCell>Row two col ones</TableCell>
    <TableCell>Row two col two</TableCell>
    <TableCell colspanForCell="2">Row two col three</TableCell>
  </TableRow>
  <TableRow selectRow="true">
    <TableCell>http://www.cnn.com</TableCell>
    <TableCell>Row 3 col two</TableCell>
    <TableCell>Row 3 col three</TableCell>
    <TableCell useRenderer="true">
      <![CDATA[<div style="font-size:x-small">
        <a href="http://www.google.com">google</a>
      </div>]]>
    </TableCell>
  </TableRow>

```


Result



3.1.2.1.3.2 RSS Transformer

Transforms RSS documents to XHTMLB.

Attribute	Value
Name	RSS_TO_XHTMLB
Version	1.0
From scheme	RSS URI: http://purl.org/rss/1.0/ With Dublin-Core URI: http://purl.org/dc/elements/1.1/ Definitions of items and in RDF format of RSS feeds URI: http://www.w3.org/1999/02/22-rdf-syntax-ns# .
To scheme	XHTMLB

Parameters

Following parameters are available:

Parameter	Type (Default value)	Description
LinksTarget	String (<i>_RSSItemWindow</i>)	Define the target frame of the links in the RSS.
ShowDatesMode	String (<i>ITEMS_ONLY</i>)	Define the type of dates that will be rendered. Possible values are: <ul style="list-style-type: none"> MAIN_ONLY Only the feed update date will be shown.

		<ul style="list-style-type: none"> ALL All dates will be shown ITEMS_ONLY Only the item dates will be shown.
ShowTime	xs:boolean (<i>true</i>)	<p>Specified if the time of every item should be displayed.</p> <p>The date displayed is affected by the <code>ShowDatesMode</code> parameter.</p>
GroupByDate	xs:boolean (<i>true</i>)	<p>Specified if groups according to the item dates should be created.</p> <p>When <code>ShowDatesMode</code> is set to <code>MAIN_ONLY</code> this parameter has no affect.</p>
DateFormat	String (<i>EEEE, MMM dd, yyyy</i>)	<p>Defines the output format of the date. The syntax is defined in <code>java.text.SimpleDateFormat</code> class documentation of the JDK.</p> <p>The actual formatting is done using the object supplied in by <code>xslDateFormatter</code> parameter.</p>
TimeFormat	String (<i>HH:mm:ss a</i>)	<p>Defines the output format of the time. The syntax is defined in the <code>java.text.SimpleDateFormat</code> class documentation of the JDK.</p> <p>The actual formatting is done using the object supplied in by <code>xslDateFormatter</code> parameter.</p>
xslDateFormatter	com.sap.portal.httpconnectivity.transformationservice.xslextensions.XSLDateFormatter (<i>an instance of the class</i>)	A parameter holding the class that formats the input date string according to the output date/time formats.
ScrollHeight	Integer (<i>300</i>)	The height of the scroll area of the items list.

3.1.2.1.3.3 Busdoc Transformer

Transforms Busdoc XML files to XHTMLB. The transformer supports all the data set types (HRField, HRScript, HRFreeText and HRRow). It also supports navigation within the table view of the data. HRNP links and the Drag & Relate feature are not supported.

Attribute	Value
Name	BUSDOC_TO_HTMLB
Version	1.0
From scheme	Busdoc

To scheme	XHTMLB
-----------	--------

Parameters

Following parameters are available:

Parameter	Type (Default value)	Description
RenderDataAsHtml	Boolean (<i>true</i>)	Specifies if the table data is rendered as HTML or plain data.
VisibleRowCount	Integer (<i>10</i>)	Specifies the number of visible rows in the table.

3.1.2.1.4 XSLDateFormatter

The XSLDateFormatter is an extension to the XSL standard. It converts date and time into a defined output format.

XSLDateFormatter class name:

```
com.sap.portal.httpconnectivity.transformationservice.xslextensions.XSLDateFormatter
```

The class has following method:

```
public static String formatDate(String date, String format)
```

It formats the specified `date` according to the specified parameter. The date format follow the specifications of class `java.text.SimpleDateFormat` that can be found in JDK documentation. The specified date must have one of the following formats:

- “EEE, dd MMM yyyy kk:mm:ss z”
Example: Wed, 03 Dec 2003 07:10:05 GMT
- “yyyy-MM-dd'T'kk:mm:ss:SS”
Example: 2004-02-03T20:53-08:00
- “yyyy-MM-dd'T'kk:mm:ss”
Example: 2004-01-12T01:37:54-400

Example

This example is taken from the RSS_TO_HTMLB transformer and is used to normalize the date formats.

```
<?xml version="1.0"?>
<xsl:stylesheet
  version="1.0" ...
  xmlns:transDate="com.sap.portal.httpconnectivity.transformationsservice.
                    xslextensions.XSLDateFormatter"
>
...
<xsl:template match="*" mode="EPRSS:Date">
  <xsl:variable name="DateStr"><xsl:value-of select="."/></xsl:variable>
  <xsl:value-of name="formatted"
    select="transDate:formatDate($DateStr,$TimeFormat)"/>
</xsl:template>
...
```

3.1.2.2 Displaying External XML-Based Content

This section describes how to integrate and display XML-based third-party content in the portal.



This section is intended for third-party content providers who want to create business packages of iViews that can display their content in the portal.

Purpose

Third-party content providers supply a variety of information, such as news headlines, weather reports, market data and industry-specific reports. Generally, this content is available on the web in XML format via simple HTTP requests or SOAP messages.

The portal's content provider framework enables you to easily create business packages of iViews, pages and worksets that will display this content in the portal. The framework works with both HTTP requests and SOAP messages, and helps you either create the proper URL or SOAP message for each XML source.

The portal already provides an XML iView template that enables you to specify an XML source URL and a transformer. The XML iView can then render the content. The framework extends this capability by providing:

- Portal components that transform and render XML content retrieved via SOAP
- Dynamic creation of URLs for content retrieved via HTTP
- Administration and personalization of parameters
- Linking between content provider iViews
- Dynamic authentication parameters, such as user name and password
- Basic Drag&Relate

This section describes the following:

- [Architecture \[Page 33\]](#): Provides background information on the content provider framework.
- [How to Build a Content Provider \[Page 33\]](#): Describes how to use the content provider framework API to build a content provider.

Creating and Managing Content

- [Essential Information \[Page 33\]](#): The portalapp.xml configuration and the JAR files for working with the content provider framework.

Implementation Considerations

For content providers with simple XML sources retrieved via HTTP requests that do not need to set parameters at runtime, it may be simpler to use XML iViews.

For more information, see the SAP NetWeaver documentation on the Help Portal (<http://help.sap.com>) → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *Content Administration* → *iViews* → *Creating iViews* → [Creating XML iViews \[External\]](#).

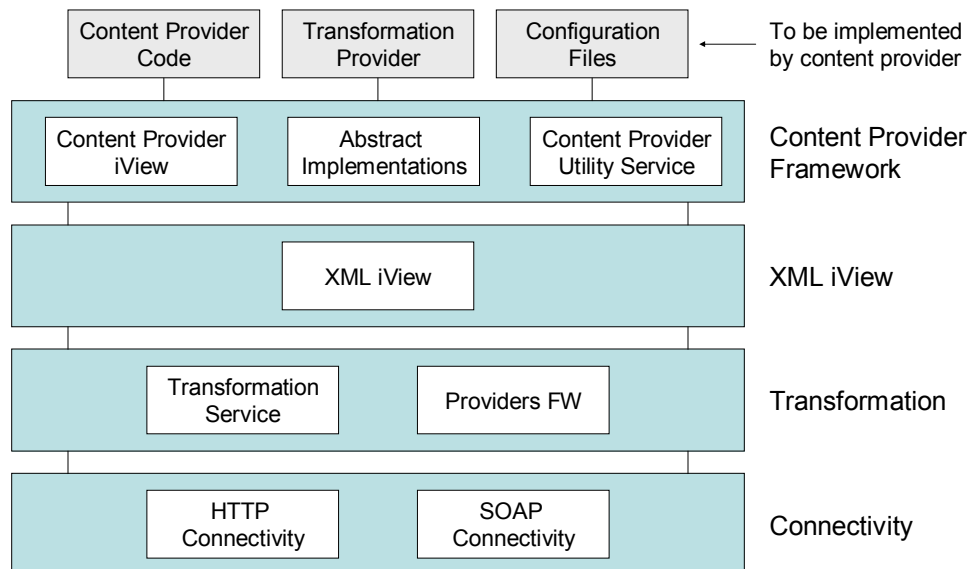
Constraints

With the content provider framework, headers cannot be set using HTTP connectivity. Headers can be set with SOAP connectivity.

There is no default mechanism for caching SOAP responses.

3.1.2.2.1 Architecture

The content provider architecture is composed of the following layers:



- **Connectivity:** Built-in portal and J2EE services that provide HTTP and SOAP connectivity.
- **Transformation:** A built-in portal service that enables XML transformations. The portal provides several generic transformers. This layer provides a framework to enable developers to package and deploy their own transformers, either XSL stylesheets or SAX handler classes.
- **XML iView:** A built-in portal component that transforms and renders XML content with the help of the transformation service.
- **Content Provider Framework:** A built-in set of portal classes that enables content providers to write code to render their content in the portal. The framework provides the following components:

Creating and Managing Content

- **Content Provider iView:** A portal component, based on the XML iView component, that can render XML content in the portal.
- **Abstract Implementations:** A set of classes that provides default implementations for most of the classes that a content provider must implement.
- **Content Provider Utility Service:** A service that helps create or initialize helper objects for a default implementation of a content provider. For example, this service can create a default source properties handler, the object that manages parameters for HTTP and SOAP requests for XML sources.

For information on this service, see [Content Provider Utility Service \[Page 33\]](#).

- **Proprietary Content Provider Implementation:** The code that a content provider must write in order to create iViews that display its content.

3.1.2.2.1.1 Content Provider Objects

To display XML content in the portal using the content provider framework, you must create the following content provider objects:

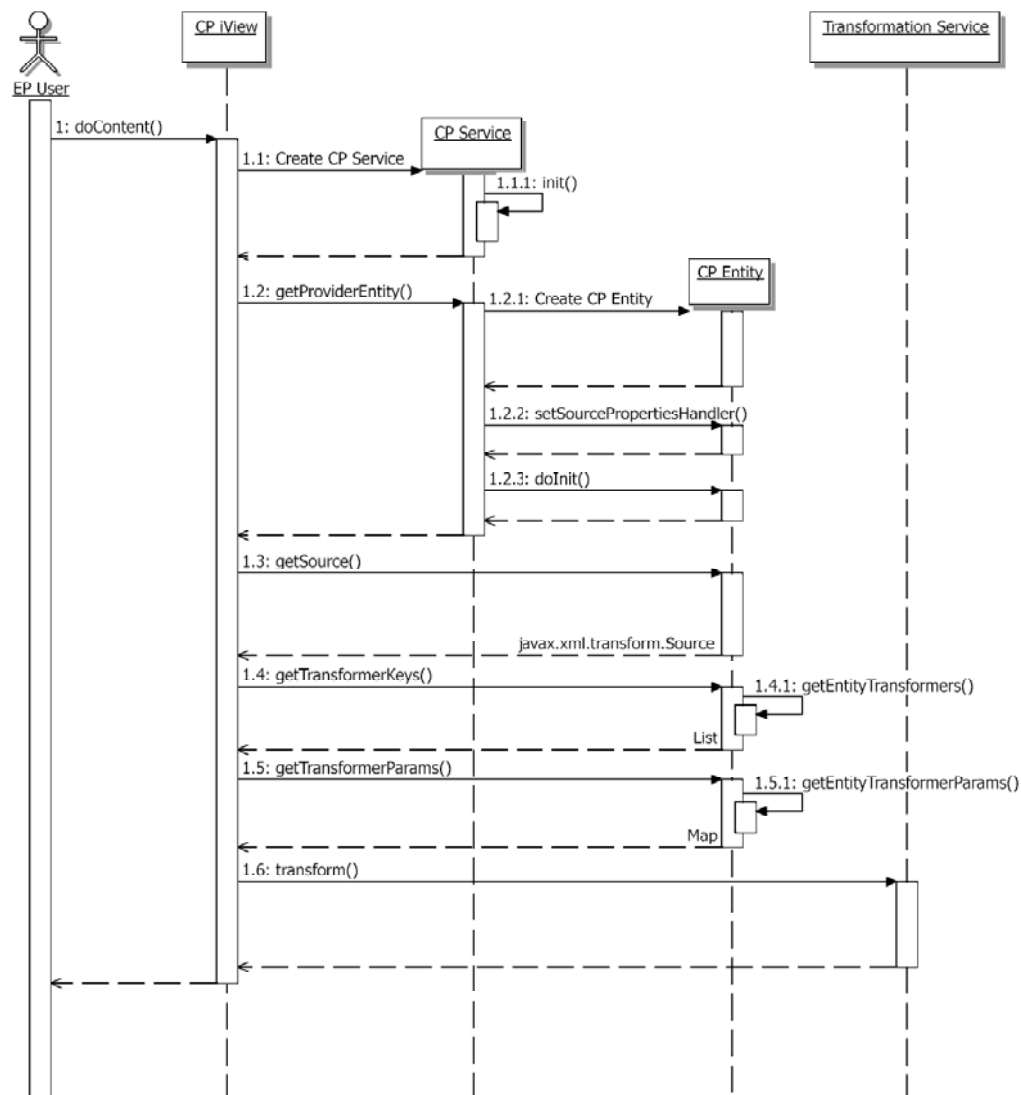
- **Service:** Represents a content provider, and is responsible for initializing the connection, creating entities and authentication.
- **Entity:** Represents a single XML source from a content provider. A content provider generally provides a number of XML sources, for example, one for news headlines and another for weather.
- **Authentication Manager (HTTP only):** Handles HTTP authentication parameters.
- **SOAP Message Processor (SOAP only):** Processes SOAP request messages just before they are sent and response messages just after they are received. The processor can also modify a response message when an error occurs.
- **Portal Component:** Requests XML from an entity, and then transforms and renders the content.

You must also provide, in your PAR file, XML transformers for each XML source. For more information, see [Transformation Service \[Page 33\]](#).

3.1.2.2.1.2 What Happens During Runtime?

The following diagram describes the runtime process that occurs when a user selects a content provider iView, triggering a call to the `doContent()` method of the iView's portal component:

Creating and Managing Content



The following describes the major steps shown in the diagram above:

1. A content provider iView obtains an instance of the content provider service.

The provider service is specified by the `com.sap.portal.cp.CP_SERVICE_KEY` property for the iView's portal component in `portalapp.xml`.

- a. The first time the service is loaded, the service's `init()` method is called (like all services). This method can be used to initialize authentication, check the HTTP connection or load a license.

2. The content provider iView calls the content provider service's `getProviderEntity()` method in order to get the provider entity for this iView.

The content provider iView sends as a parameter the type of entity, which is equal to the `com.sap.portal.cp.CP_ENTITY_KEY` property for the iView's portal component in `portalapp.xml`.

Creating and Managing Content

3. The content provider service obtains an instance of the appropriate content provider entity, calls the entity's `setSourcePropertiesHandler()` and `doInit()` methods, and returns the entity to the content provider iView.

`setSourcePropertiesHandler()` sets the entity's properties handler (`ISourcePropertiesHandler`), which determines how to retrieve parameter values for use with the HTTP or SOAP request. Generally, you can create the default implementation of the `ISourcePropertiesHandler` object with the help of the content provider utility service.

`doInit()` performs any necessary initialization for the entity, including setting the entity's request object.

4. The content provider iView calls the entity's `getSource()` method, which returns a `javax.xml.transform.Source` object. This object contains the information needed for making the request to the content provider and getting an XML source.

This is the key step for retrieving the XML source. The required tasks performed by this method are different for HTTP and SOAP entities, and are explained in more detail in [getSource\(\) Method \(HTTP\) \[Page 33\]](#) and [getSource\(\) Method \(SOAP\) \[Page 33\]](#).

5. The content provider iView calls the entity's `getTransformerKeys()` and `getEntityTransformerParams()` methods to get information about the transformation provider and transformer for this entity.

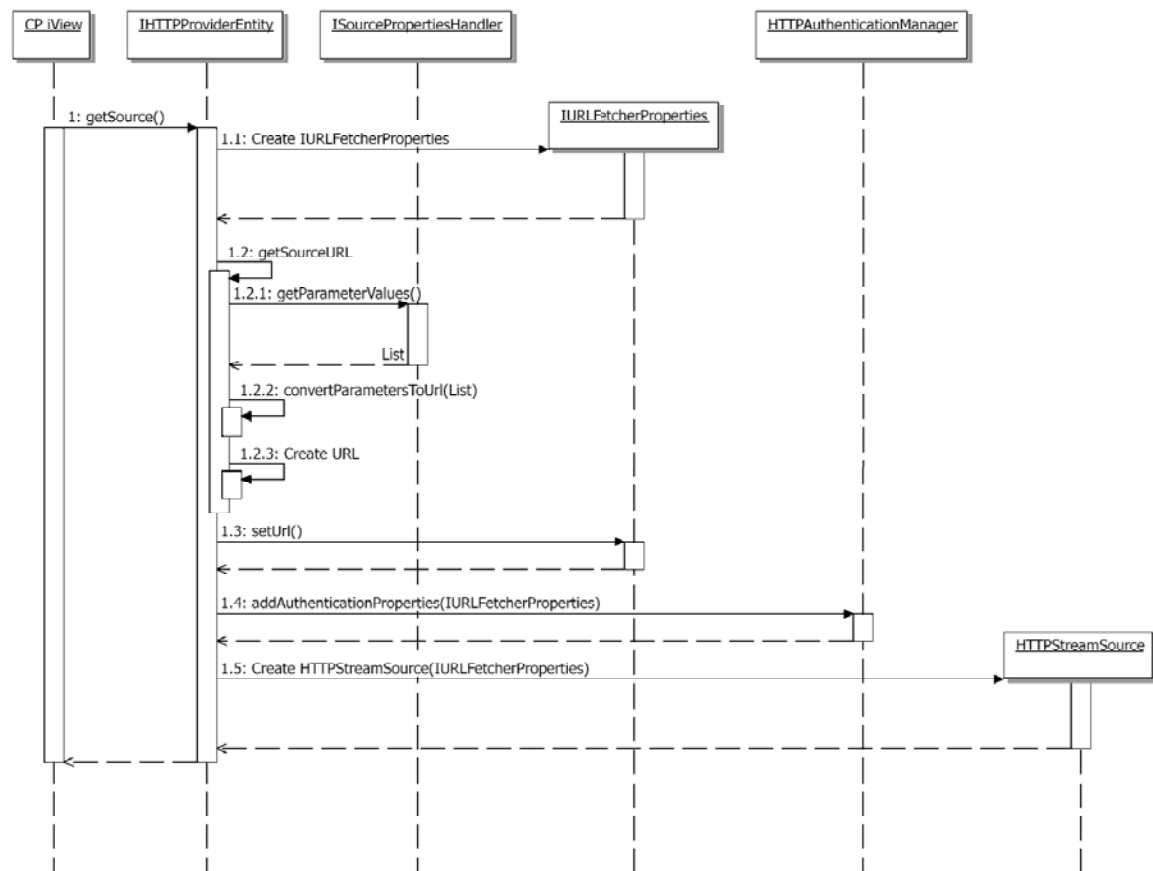
The transformer for the entity is specified by the `com.sap.portal.cp.TRANSFORMER_NAME` property for the iView's portal component in `portalapp.xml`.

6. The content provider iView transforms the XML and renders the content.

3.1.2.2.1.2.1 **getSource() Method (HTTP)**

The following diagram describes the runtime process that occurs in the `getSource()` method of HTTP entities:

Creating and Managing Content

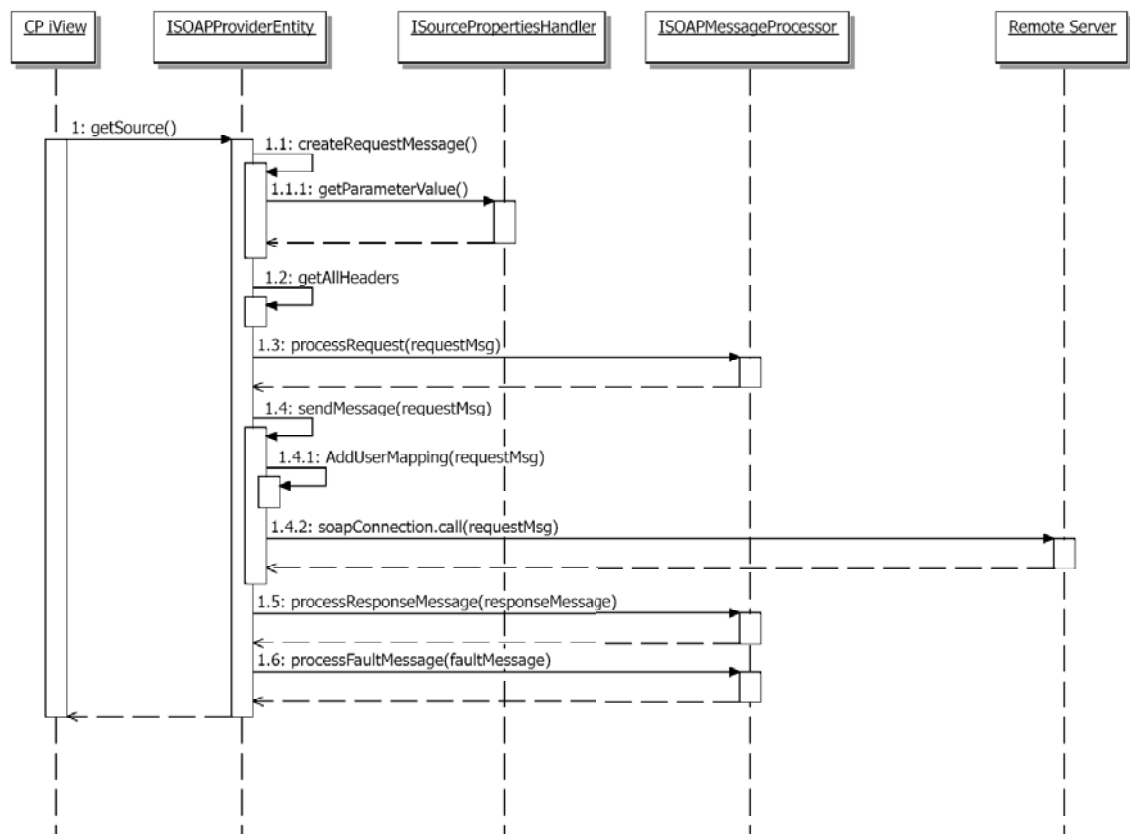


In the above process, you are required only to implement `getSourceURL()`, which creates the URL for retrieving the XML content.

3.1.2.2.1.2.2 `getSource()` Method (SOAP)

The following diagram describes the runtime process that occurs in the `getSource()` method of SOAP entities:

Creating and Managing Content



In the above process, you are required only to implement `createRequestMessage()`, which creates the SOAP message for retrieving the XML content.



The `processFaultMessage()` is called only if a fault occurs.

3.1.2.2.1.3 Parameter Handling

One of the key features of the content provider framework is that it simplifies the setting of parameters that are used to create the HTTP or SOAP request that retrieves the XML content.

When a parameter can be retrieved from several sources, a mechanism must be created to retrieve the value from the correct location. The `ISourcePropertiesHandler` interface provides this mechanism. Each HTTP and SOAP entity is assigned an `ISourcePropertiesHandler` object to handle the retrieval of its parameters.

You can create an `ISourcePropertiesHandler` object with the default implementation with the help of the content provider utility service. This default implementation searches for parameter values in the following order:

1. Drag&Relate value
2. Request query string
3. Page Context

Creating and Managing Content

4. Personalized value
5. Value defined in iView
6. Value defined in portal component (portalapp.xml)

For more information on where these parameters are set and how the parameters are retrieved, see [Parameters and Constants \[Page 33\]](#).

3.1.2.2.1.4 Linking iViews

The content provider framework helps you create links between your content provider iViews. To create a link from one iView to another, pass to the transformer of your first iView the URL of the portal component of your second iView. The transformer can then add parameters to the URL, for example, a story ID, and create the complete link.

You can create the URL to a portal component with the following code:

```
IPortalComponentURI componentURI =
request.createPortalComponentURI();

componentURI.setContextName(m_serviceContext.getApplicationName()
    + "." + "NameOfPortalComponent");

myUrl = componentURI.toString();
```

In the example above, the first line creates an `IPortalComponentURI` object from the request object.

The second line gets the URL for the portal component called `NameOfPortalComponent` located within the PAR that contains your content provider service. This PAR should also contain all your content provider portal components.

The third line converts the URL to a string and stores it in the variable `myURL`, which you can pass to the transformer for the current entity. To pass parameters to an entity's transformer, implement the `getGeneralParameters()` or `getEntityTransformerParams()` method in the entity class.

3.1.2.2.1.5 Content Provider Utility Service

The content provider framework provides a utility for creating helper objects that are required by the default implementation of a content provider. The following code retrieves the service:

```
IContentProvidersUtilsService utilsService =
    (IContentProvidersUtilsService)
    m_context.getService(IContentProvidersUtilsService.KEY);
```

You can also retrieve the utility service by calling the `getUtilsService()` method of the `AbstractContentProviderService` class.

The utility service can create instances of the default implementation of the following interfaces:

- **ISourcePropertiesHandler:** Retrieves property values, and sets the order of lookup when the value can be obtained from several sources.
- **IContextWrapper:** Handles events in the HTMLB page context for the current request. This object is used for creating a source properties handler.
- **IDataHandler:** Handles properties for an entity's corresponding portal component. This object is used for creating a source properties handler.

Creating and Managing Content

- **IProviderDetails:** Holds information about the content provider service. The utility creates an empty object.

3.1.2.2 How to Build a Content Provider

A content provider is a PAR with the following parts:

- **Content Provider Service:** A Java class that implements `IContentProviderService`, whose main task is to create the entities for this content provider.

For HTTP connectivity, you may also need to create a class that implements `IHTTPAuthenticationManager` in order to provide authentication information (for example, user name and password) for the connection.

For SOAP connectivity, you may also need to create a class that implements `ISOAPMessageProcessor` in order to process SOAP messages before they are sent and after they are received.
- **Content Provider Entity:** A Java class that implements `IContentProviderEntity` and `ITransformationProviderEntity`, whose main task is to create the URL for the HTTP request or the message for the SOAP request that retrieves the XML source. A content provider generally has many entities, and one Java class may provide the implementation for more than one entity.



You can create HTTP and SOAP entities for the same provider.

- **Entity Transformer:** An XSL file or SAX handler class that transforms the XML for rendering in an iView. Each entity must be assigned a transformer.
- **Entity Portal Component:** A portal component that can issue a HTTP or SOAP request, retrieve an XML source, transform the XML and render the information in the portal. Each entity must be assigned to an entity portal component.

You do not have to write any Java code to create an entity portal component. Your portal components are simply derived from the portal component defined in `com.sap.portal.contentproviders.runtime.ContentProvidersComponent`, and you create them simply by adding a `<component>` element in `portalapp.xml`.

The PAR file can contain any number of entities, transformers and portal components. However, for each XML data source provided by the content provider, you must assign one entity, one transformer and one component.

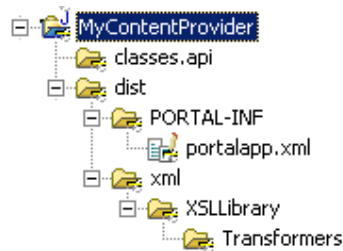
Workflow

The following are the steps required for creating a content provider:

1. [Create a Content Provider Service \[Page 33\]](#)
2. [Create Content Provider Entities \[Page 33\]](#)
3. [Create Entity Portal Components \[Page 33\]](#)
4. [Create Entity Transformers \[Page 33\]](#)
5. [Create a Business Package \(.sda file\) \[Page 33\]](#)

Prerequisites

- You have created a skeleton portal application project, with at least the following folders and files:



In Developer Studio, you can create the project in the usual way: Select *New* → *Project*, and then select *Portal Application* → *Create a Portal Application Project*. After the wizard creates the basic skeleton, add the folders that will hold the XML transformers (the xml folder and subfolders), as shown above.

For more information on the directory structure required for XML transformers, see [Transformation Service \[Page 33\]](#).

- You understand how to create XML transformers. The XML transformers that are to be used with this content provider implementation are defined in the same PAR that defines the content provider implementation.

For more information, see [Transformation Service \[Page 33\]](#).

- You understand how to create SOAP requests. The content provider framework uses standard Java classes for handling SOAP.
- The third-party content is in XML format and is exposed via the web, either via HTTP or SOAP requests.



The content provider framework is shipped with the portal as of SAP Enterprise Portal SP Stack 12.

3.1.2.2.1 Step 1: Creating a Content Provider Service

The content provider service represents the third-party content provider. This service is responsible for initializing the connection to the provider, defining authentication parameters, and creating provider entities.

All content provider services implement `IContentProviderService`. The content provider framework includes an `AbstractContentProviderService` class, which implements all required methods except for `getProviderEntity()`, which you must implement.

In the following procedure, the variable `context` is the generic `IServiceContext` object for the current service.

Procedure

- Create a new class that extends `AbstractContentProviderService`.
- If necessary, override the provider service's `init()` method in order to handle such tasks as initializing authentication, checking the connection or loading a license.

Creating and Managing Content

For example, the following creates a new HTTP authentication manager and stores it in a local variable in the service class. The authentication manager is then available to be assigned to the entity created in the `getProviderEntity()` method.

```
super.init(context);
mm_authManger = new myCPAuthenticationManager(context);
```

For more information on authentication managers, see [Creating an Authentication Manager \[Page 33\]](#).

3. Implement the `getProviderEntity()` method, which serves as a factory for all entity objects for this content provider.

In this method, do the following:

- Create an `IDataHandler` object with the help of the content provider utility service, which you can obtain from the `getUtilsService()` method of the `AbstractContentProviderService` class.

```
IDataHandler dataHandler = getUtilsService().createDataHandler(
    request, context);
```

- Obtain (or, if necessary, create) the appropriate entity based on the `ProviderEntityType` parameter passed into the method.

In the following implementation, an instance of `myCPSearchEntity` is created if the entity type is `SEARCH`. Otherwise, an instance of `myCPEntity` is created. (`myCPEntity` can be the implementation for different entities because, with the same `getSourceURL()` method, it can return different URL strings based on the entity type).

```
myCPEntity res = null;
if (providerEntityType.equals(ProviderEntityType.SEARCH))
{
    res = new myCPSearchEntity(m_context, dataHandler);
}
else if (providerEntityType.equals(ProviderEntityType.STORY)
    || providerEntityType.equals(ProviderEntityType.COMPANY)
    || providerEntityType.equals(ProviderEntityType.HEADLINE)
    || providerEntityType.getValue().equals(IConstants.STOCK_ENTITY_KEY)
    || providerEntityType.getValue().equals(IConstants.CHARTS_ENTITY_KEY)
    || providerEntityType.getValue().equals(IConstants.FINANCE_ENTITY_KEY)
    || providerEntityType.getValue().equals(IConstants.WEATHER_ENTITY_KEY)
    || providerEntityType.getValue().equals(IConstants.FORECAST_ENTITY_KEY))
{
    res = new myCPEntity(providerEntityType, m_context, dataHandler);
}

if (res == null)
{
    throw new ContentProviderException("Cannot
        create provider entity: "+providerEntityType);
}
```

- Create a default properties handler and make this object the properties handler for the entity.

```
IContextWrapper myCPContextHandler =
    getUtilsService().createContextWrapper(context);
```

Creating and Managing Content

```

ISourcePropertiesHandler propsHndlr =
    getUtilsService().createSourcePropertiesHandler(
        request, dataHandler, myCPContextHandler);

res.setSourcePropertiesHandler(propsHndlr);

```

- For HTTP connectivity, set the authentication manager for the entity to the service's authentication manager. You should initialize the authentication manager in the provider service's `init()` method.

```
res.setHTTPAuthenticatorManager(mm_authManger);
```

- For SOAP connectivity, set the message processor for the entity. For more information on creating a SOAP message processor, see [Creating a SOAP Message Processor \[External\]](#).

```
res.setMessageProcessor( new MyMessageProcessor());
```

- Call the entity's `doInit()` method, and then return the entity.

```
res.doInit(request);
return res;
```

4. Create a service entry in `portalapp.xml` for the provider service, similar to the one shown below:

```

<service name="ContentProvider">
  <service-config>
    <property name="className" value="com.sap.portal.myCP.myCPService"/>
    <property name="startup" value="false"/>
    <property name="SafetyLevel" value="high_safety"/>
  </service-config>
  <service-profile>
    <property name="com.sap.portal.cp.PROVIDER_SITE"
      value="http://www.myCP.com"/>
    <property name="com.sap.portal.cp.PROVIDER_ICON"
      value="http://www.myCP.com/logobar.gif"/>
    <property name="com.sap.portal.cp.PROVIDER_VERSION" value="1.0"/>
    <property name="com.sap.portal.cp.PROVIDER_NAME" value="myCP"/>
    <property name="com.sap.portal.cp.BASE_URL"
      value="http://myCP.com/api"/>
  </service-profile>
</service>

```

The `<service>` element for the content provider service is similar to the `<service>` element for any service. It defines the name of the service in the element's `name` attribute.

In the `<service>` element, add a `<service-config>` element, as you would for any service. In this element, add the following property elements:

- `className`: The name of your class that implements the provider service.
- `startup`: Set to `false`, indicating not to start this service at portal startup.
- `SafetyLevel`: Set to `high_safety`. This is the service's security zone safety level.

In the `<service>` element, also add a `<service-profile>` element, as you would for any service. In this section, add the following property elements:

- Information about the content provider service, as defined in the `IProviderDetails` interface:
 - `com.sap.portal.cp.PROVIDER_SITE`: The main web site of the content provider

Creating and Managing Content

- `com.sap.portal.cp.PROVIDER_ICON`: The URL of the content provider's logo
- `com.sap.portal.cp.PROVIDER_VERSION`: The version of the content provider
- `com.sap.portal.cp.PROVIDER_NAME`: The name of the content provider

The default implementation of the content provider framework currently does not make use of these properties, and you are not required to supply them in `portalapp.xml`.

- `com.sap.portal.cp.BASE_URL`: The base address for all XML sources for this content provider. You do not need to use this address; however, the default implementation loads this value into the entity's `m_baseUrl` field during initialization.
- Any other properties that the content provider needs, for example, a default user name and password in order to connect.

3.1.2.2.1.1 Creating an Authentication Manager

An authentication manager supplies the user name and password, as well as any other parameters, for authentication during HTTP requests to the content provider.



An authentication manager is for HTTP connectivity only.

The content provider service creates an instance of the authentication manager, and supplies it to each entity. During initialization of a HTTP connection, the entity creates an `IURLFetcherProperties` object and passes it to its authentication manager, which loads the `IURLFetcherProperties` object with the authentication parameters. The entity later provides the parameters to the HTTP connectivity service.

Procedure

1. Create a class that implements `IHTTPAuthenticationManager`.
2. Implement the `addAuthenticationProperties()` method. In this method, supply parameter key-value pairs to the `IURLFetcherProperties` object passed into the method, as in the following example:

```
public class myAuthenticationManager implements IHTTPAuthenticationManager
{
    private String mm_password;
    private String mm_userName;

    public myAuthenticationManager(IServiceContext m_context)
    {
        super();
        mm_userName = m_context.getServiceProfile().getProperty(
            myContentProviderService.IConstants.USER_NAME);
        mm_password = m_context.getServiceProfile().getProperty(
            myProviderService.IConstants.PASSWORD);
    }

    public void addAuthenticationProperties(IURLFetcherProperties
properties)
        throws ContentProviderException
    {
```


Creating and Managing Content

```

    try
    {
        properties.getParameters().addParameter("id",mm_userName);
        properties.getParameters().addParameter("password",mm_password);
    }
    catch (InvalidParameterException e)
    {
        throw new ContentProviderException(
            "Unable to set authentication parameters",e);
    }
}
}

```

In the example above, the authentication manager's constructor retrieves the user name and password from property values set for the content provider service in `portalapp.xml`. Note that the `IServiceContext` passed into the constructor is that for the provider service.

If necessary, add property elements in `portalapp.xml` to store the user name and password.

3.1.2.2.2.2 Step 2: Creating Content Provider Entities

A content provider entity represents one XML source from a specific content provider. The entity provides all the information required for making a connection with the content provider and retrieving the specific XML source.

Several entities can be implemented by the same Java class.

There are two types of entities:

- **HTTP Entity:** An entity that obtains XML via an HTTP request. For information on creating HTTP entities, see [Creating HTTP Entities \[Page 33\]](#).
- **SOAP Entity:** An entity that obtains XML via a SOAP request. For information on creating SOAP entities, see [Creating SOAP Entities \[Page 33\]](#).

All entities implement the following interfaces:

- **IProviderEntity:** Defines the methods for getting the XML source.
- **ITransformationProviderEntity:** Defines the methods for getting information about the entity's transformer.

3.1.2.2.2.2.1 Creating HTTP Entities

The portal includes an `AbstractHTTPProviderEntity` class that implements all required methods of `IProviderEntity` and `ITransformationProviderEntity` interfaces in order to make a HTTP request.

You need, however, to implement the `getSourceURL()` method (from the `IHTTPProviderEntity` interface, which `AbstractHTTPProviderEntity` also implements). `getSourceURL()` returns the URL string for the HTTP request for the XML source.

Procedure

1. Create a class that extends `AbstractHTTPProviderEntity`.
2. Implement the `getSourceURL()` method, which should return the URL for the HTTP request to the XML source. The following is an example:

```

public String getSourceURL() throws ContentProviderException

```

Creating and Managing Content

```

{
    List params =
        m_sourcePropertiesHandler.getParametersValues(m_request);

    String staticUrlPart = (String) m_dataHandler.getProperty("Static");

    return m_baseUrl + staticUrlPart + "&" +
        this.convertParametersToUrl(params);
}

```

- **Base URL:** The base URL is taken from the entity's `m_baseUrl` field, which the default implementation loads with the value set in the service's `com.sap.portal.cp.BASE_URL` property, which is defined in `portalapp.xml`.
- **Static URL:** The static part of the URL is taken from the value set in the entity's `Static` property, which is defined in `portalapp.xml`. This property is not part of the default implementation.
- **Parameters:** The URL parameters are stored in the entity's source properties handler (which was defined when the entity was retrieved by the provider service in its `getProviderEntity()` method).

These URL parameters are listed in the `com.sap.portal.cp.PARAMETERS_LIST` property in the `<component>` element for the entity in `portalapp.xml`. The default values are defined in separate `<property>` elements for each parameter. For more information on these properties, see [Parameters and Constants \[Page 33\]](#).

3. If necessary, override the `doInit()` method.
4. If necessary, override the `getGeneralParameters()` method. This method defines parameters that are passed to all the transformers associated with this entity.
5. If necessary, override the `getEntityTransformerParams()` method. This method can be used to pass different key-value pairs to different transformers associated with this entity. You can determine the current transformer by examining the `ITransformerInformation` object passed into the method.

3.1.2.2.2.2 Creating SOAP Entities

The portal includes an `AbstractSOAPProviderEntity` class that implements all required methods of `IProviderEntity` and `ITransformationProviderEntity` interfaces in order to make SOAP requests.

You need, however, to implement the `createRequestMessage()` method, which returns the SOAP message as defined in the content provider's WSDL.

Procedure

1. Create a class that extends `AbstractSOAPProviderEntity`.
2. Implement the `createMessage()` method, which should return a SOAP message for retrieving the XML source. The following is an example:

```

protected SOAPMessage createRequestMessage()
    throws ContentProviderException, SOAPException
{
    // Create SOAP Envelope
    SOAPMessage msg = createBasicEnvelope();
    SOAPEnvelope env = msg.getSOAPPart().getEnvelope();
}

```

Creating and Managing Content

```

SOAPBody body = env.getBody();

// Create GetNewsstandHeadlines Element
SOAPElement getNewsStandHlElement = body.addChildElement(
    env.createName("GetNewsstandHeadlines", "",
        ICPService.CP_NS_DEV2_1_PARSERS));

// Create sectionIDs Element
SOAPElement sectionIdsElm =
    getNewsStandHlElement.addChildElement(
        env.createName("sectionIDs", "",
            ICPService.CP_NS_DEV2_1_PARSERS));

// Create sectionID Element
SOAPElement sectionIDElm = sectionIdsElm.addChildElement(
    env.createName("sectionID", "",
        ICPService.CP_NS_DEV2_1_PARSERS));

// Create Section Code
sectionIDElm.addTextNode(
    m_sourcePropertiesHandler.getParameterValue(
        m_request, "sectionCode"));

return msg;
}

```

3. If necessary, override the `doInit()` method.
4. If necessary, override the `getGeneralParameters()` method. This method defines parameters that are passed to all the transformers associated with this entity.
5. If necessary, override the `getEntityTransformerParams()` method. This method can be used to pass different key-value pairs to different transformers associated with this entity. You can determine the current transformer by examining the `ITransformerInformation` object passed into the method.

3.1.2.2.3 Step 3: Creating Entity Portal Components

Each content provider entity needs a component that can render its content in an `iView`. The content provider framework includes the `ContentProvidersComponent` class on which you can base your content provider portal components.

Procedure

1. Add an entry in `portalapp.xml` that creates a portal component based on the `ContentProvidersComponent` class.



The portal component is based on `com.sap.portal.contentproviders.runtime.ContentProvidersComponent` and you do not need to write any code for this component.

The following is an example of an entry in `portalapp.xml` for an entity portal component:

```

<component name="CP_STORY">
  <component-config>

```

Creating and Managing Content

```

    <property name="ClassName" value="com.sap.portal.contentproviders.
runtime.ContentProvidersComponent"/>
    <property name="SafetyLevel" value="low_safety"/>
  </component-config>
  <component-profile>
    <property name="com.sap.portal.cp.CP_SERVICE_KEY"

        value="com.sap.portal.myCP.ContentProvider"/>
    <property name="com.sap.portal.cp.ENTITY_KEY" value="CP_STORY"/>
    <property name="com.sap.portal.cp.TRANSFORMER_NAME"
value="myTrans"/>
    <property name="com.sap.portal.cp.PARAMETERS_LIST"
value="story_id"/>
    <property name="com.sap.portal.cp.parameter.story_id" value="">
      <property name="personalization" value="dialog"/>
      <property name="plainDescription" value="Story ID"/>
    </property>
  </component-profile>
</component>

```

The `<component>` element is similar to the `<component>` element for any portal component. It defines the name of the service in the element's name attribute.

In the `<component>` element, add a `<component-config>` element, as you would for any component. In this element, add the following property elements:

- `className`: The name of the class that implements the component.
- `SafetyLevel`: The component's security zone safety level, which should be set to `low_safety`.

In the `<component>` element, add a `<component-profile>` element, and include the following property elements:

- `com.sap.portal.cp.CP_SERVICE_KEY`: The provider service that the component instantiates to start the retrieval of XML content.
- `com.sap.portal.cp.ENTITY_KEY`: A value representing the entity to be used for retrieving the XML content for this component. This value is passed into the provider service's `getProviderEntity()` method, which can use the value to determine which entity to return.
- `com.sap.portal.cp.TRANSFORMER_NAME`: The name of the transformer the component uses to transform and render the XML.
- `com.sap.portal.cp.PARAMETERS_LIST`: A list of parameters for the HTTP or SOAP request. The default value for each parameter is defined in another property element whose name is a concatenation of `com.sap.portal.cp.parameter`, a period (.) and the name of the parameter.

For example, the default value for a parameter named `ticker` is defined in a property element named `com.sap.portal.cp.parameter.ticker`.

- `com.sap.portal.cp.parameter.<parameterName>`: The default value for any parameter defined in the `com.sap.portal.cp.PARAMETERS_LIST` property element.

3.1.2.2.2.4 Step 4: Creating Transformers

For each XML source, you must create a transformer in order to render the XML data in the content provider iView. Package the transformers in the same PAR in which you package the content provider.

For information on creating transformers, see [Transformation Service \[Page 33\]](#).

Assign each transformer to its corresponding entity by adding a `com.sap.portal.cp.TRANSFORMER_NAME` property for the portal component created for the entity. The value of this property should be the name of the transformer for the entity, which is listed in the `Name` property for the transformer in the `Transformers.xml` file.

For more information on assigning a transformer to an entity's portal component, see [Creating Entity Portal Components \[Page 33\]](#).

3.1.2.2.2.5 Step 5: Creating a Content Provider Business Package

After you create the content provider classes and configuration files, create a PAR file from them and then a transport package (`.epa` file) of iViews, pages and worksets based on your portal components. Finally, package these files into a business package (`.sda` file) for delivery.

Procedure

1. Compile and package your content provider components into a PAR file.
2. Create iViews, pages and worksets based on your content provider portal components.
For more information on creating iViews, Pages and Worksets, go to the Help Portal at <http://help.sap.com> and select *Documentation* → *SAP NetWeaver*. From the documentation, see *SAP Library* → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *Content Administration* → *iViews*.
3. Export your iViews, pages and worksets into a transport package (`.epa` file).
For more information on creating iViews, pages and worksets, go to the Help Portal at <http://help.sap.com> and select *Documentation* → *SAP NetWeaver*. From the documentation, see *SAP Library* → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *System Administration* → *Transport, Upload, and Content Mirroring*.
4. Create a business package (`.sda`) from your PAR and transport package.
For more information on creating an `.sda` file, see SAP Note 696084. You can access SAP Notes on service marketplace at <http://service.sap.com/notes>.

3.1.2.2.2.6 Content Provider Logging and Tracing

The content provider framework includes the `ContentProvidersLogger` class in order to simplify writing to the J2EE log. The class exposes one static method, `getPPLogger()`, which exposes the portal platform logging interface. From this interface, you can call the `log()` and `trace()` methods to send information to the log.

All logs created with this class are written to the following category and location:

Creating and Managing Content

- **Category:** /Server
- **Location:** com.sap.portal.httpConnectivity.xmlIViewsRuntime



Writing to the log slows performance of the portal, so you should only log information when you really need to.

Type of Logging

There are two target audiences for the information sent to the log:

- **System Administration:** System administrators should be able to see where the problem occurred and what configuration problems exist.

This information should be logged using the `log()` method, as follows:

```
ContentProvidersLogger.getPPLogger().log("Information to
Log")
```

- **Development Support:** Development support engineers should be able to see what actions were performed when the problem occurred.

This information should be logged using the `trace()` method, as follows:

```
ContentProvidersLogger.getPPLogger().trace ("Information to
Log")
```

3.1.2.2.3 Essential Information

This section provides the following information to help you create a content provider:

- [portalapp.xml \[Page 33\]](#): The structure of the `portalapp.xml` file for this project.
- [JARs and Packages \[Page 33\]](#): The packages used in this project, and the JAR files required for compilation.
- [Parameters and Constants \[Page 33\]](#): A list of parameters used in this project, including where each is defined and referenced.

3.1.2.2.3.1 portalapp.xml

The following is a sample `portalapp.xml` for a content provider PAR file:

```
<?xml version="1.0" encoding="utf-8"?>
<application>
  <registry>
    <entry path="/runtime/transformers" type="subcontext"/>
    <entry path="runtime/transformers/com.sap.portal.contentproviders.myCP"
      name="TransformersProvider" type="service"/>
  </registry>
  <application-config>
    <property name="SharingReference" value="com.sap.portal.contentproviders"/>
    <property name="releasable" value="false"/>
    <property name="startup" value="true"/>
    <property name="Vendor" value="sap.com"/>
    <property name="SecurityArea" value="NetWeaver.Portal"/>
  </application-config>
  <components>
    <component name="CP_STORY">
      <component-config>
        <property name="ClassName"
          value="com.sap.portal.contentproviders.runtime.ContentProvidersComponent"/>
        <property name="SafetyLevel" value="low_safety"/>
      </component-config>
    </component>
  </components>
</application>
```

Creating and Managing Content

```

    <component-profile>
      <property name="com.sap.portal.cp.CP_SERVICE_KEY"
        value="com.sap.portal.contentproviders.myCP.ContentProvider"/>
      <property name="com.sap.portal.cp.ENTITY_KEY" value="CP STORY"/>
      <property name="com.sap.portal.cp.TRANSFORMER_NAME" value="STORY TO XHTMLB"/>
      <property name="com.sap.portal.cp.PARAMETERS_LIST" value="story_id"/>
      <property name="com.sap.portal.cp.parameter.story_id" value="">
        <property name="personalization" value="dialog"/>
        <property name="plainDescription" value="Story ID"/>
      </property>
    </component-profile>
  </component>
</components>
<services>
  <service name="ContentProvider">
    <service-config>
      <property name="className"
        value="com.sap.portal.contentproviders.myCP.myCPService"/>
      <property name="startup" value="false"/>
    </service-config>
    <service-profile>
      <property name="com.sap.portal.cp.PROVIDER_SITE" value="http://www.myCP.com"/>
      <property name="com.sap.portal.cp.PROVIDER_ICON"
        value="http://www.myCP.com/logobar.gif"/>
      <property name="com.sap.portal.cp.PROVIDER_VERSION" value="1.0"/>
      <property name="com.sap.portal.cp.PROVIDER_NAME" value="myCP"/>
      <property name="PROVIDER_USERNAME" value="username"/>
      <property name="PROVIDER_PASSWORD" value="password"/>
      <property name="com.sap.portal.cp.BASE_URL"
value="http://api.myCP.com/api/?"/>
    </service-profile>
  </service>
  <service name="TransformersProvider">
    <service-config>
      <property name="className"
value="com.sap.portal.contentproviders.myCP.myCPTransformersProviderService"/>
      <property name="SafetyLevel" value="no_safety"/>
      <property name="ResourceBundleName"
        value="myCPTransformersProvider_localization"/>
      <property name="startup" value="false"/>
    </service-config>
  </service>
</services>

```

Registry Element

The `<registry>` element creates an entry in the portal registry for the transformer service that will be used for the transformers defined in the content provider framework.

For more information, see [Transformation Service \[Page 33\]](#).

```

<registry>
  <entry path="/runtime/transformers" type="subcontext"/>
  <entry path="runtime/transformers/com.sap.portal.contentproviders.myCP"
    name="TransformersProvider" type="service"/>
</registry>

```

Application Configuration Element

The `<application-config>` element is a standard element for all PAR files. In addition to the standard properties for this PAR, add a reference to `com.sap.portal.contentproviders` in the `SharingReference` property element.

```
<application-config>
```

Creating and Managing Content

```

<property name="SharingReference"
          value="com.sap.portal.contentproviders"/>
<property name="releasable" value="false"/>
<property name="startup" value="true"/>
<property name="Vendor" value="sap.com"/>
<property name="SecurityArea" value="NetWeaver.Portal"/>
</application-config>

```

Components Element

You need to create a portal component for each entity (XML source) in your project. For each portal component that you need to create, add a `<component>` element in the `<components>` section.

For more information on how to create each `<component>` section, see [Step 3: Creating Entity Portal Components \[Page 33\]](#).

```

<component name="CP_STORY">
  <component-config>
    <property name="ClassName" value="com.sap.portal.contentproviders.
runtime.ContentProvidersComponent"/>
  </component-config>
  <component-profile>
    <property name="com.sap.portal.cp.CP_SERVICE_KEY"

      value="com.sap.portal.myCP.ContentProvider"/>
    <property name="com.sap.portal.cp.ENTITY_KEY" value="CP_STORY"/>
    <property name="com.sap.portal.cp.TRANSFORMER_NAME"
value="myTrans"/>
    <property name="com.sap.portal.cp.PARAMETERS_LIST"
value="story_id"/>
    <property name="com.sap.portal.cp.parameter.story_id" value="">
      <property name="personalization" value="dialog"/>
      <property name="plainDescription" value="Story ID"/>
    </property>
  </component-profile>
</component>

```

Services Element

The `<services>` element includes two `<service>` elements, one for the content provider service and one for the transformation service.

For more information on the element for the content provider service, see [Step 1: Creating a Content Provider Service \[Page 33\]](#).

For more information on the element for the transformation service, see [Transformation Service \[Page 33\]](#).

3.1.2.2.3.2 JARs and Packages

This section lists the packages used in this project and the JAR files required for compilation.

Packages

- com.sap.portal.contentproviders

JAR Files

- com.sap.portal.contentprovidersapi.jar

3.1.2.2.3.3 Parameters and Constants

One of the main features of the content provider framework is that it helps manage all the parameters that you might want to set for your service, entities and transformers.

This section summarizes the parameters and constants used in the content provider framework:

Parameter	Where to Define	Where to Access
Entity Parameters Parameters appended to the URL for accessing the XML source for this entity.	In portalapp.xml, in a <code><property></code> element named <code>com.sap.portal.cp.PARAMETERS_LIST</code> in the entity's <code><component></code> element.	From the entity's source properties handler (<code>ISourcePropertiesHandler</code> object).
Entity Drag&Drop Parameter	In portalapp.xml, in a <code><property></code> element named <code>com.sap.portal.cp.DROP_PARAMETER_NAME</code> in the entity's <code><component></code> element The Drag&Drop parameter must be one of the entity parameters.	Accessed directly by the content provider portal component.
Entity Header Parameters For SOAP only. HTTP requests cannot include headers.	In portalapp.xml, in a <code><property></code> element named <code>com.sap.portal.cp.HEADERS_LIST</code> in the entity's <code><component></code> element	Accessed directly by the content provider portal component.
Other Entity Parameters	In portalapp.xml, in <code><property></code> element in entity's <code><component></code> element.	From the entity's data handler (<code>IDataHandler</code> object).
Content Provider Service Parameters	In portalapp.xml, in <code><property></code> element in service's <code><service></code> element.	<ul style="list-style-type: none"> • Get the service context. • Get the service profile by calling the service context's <code>getServiceProfile()</code> method. • Get the parameter by calling profile's <code>getProperty()</code> method.

Creating and Managing Content

Transformer Parameters (all transformers) Parameters for all transformers associated with an entity.	Override the entity's <code>getGeneralParameters()</code> method, which returns a <code>Map</code> object of parameter names and values. Add the required parameters to this map.	From within the XSL transformer for this entity. For example, you can define the parameter: <pre><xsl:param name="BaseUrl"/></pre> and then refer to it: <pre><xsl:value-of select="\$BaseUrl"/></pre>
Transformer Parameters (specific transformer) Parameters for a specific transformer associated with an entity.	Override the entity's <code>getEntityTransformerParams()</code> method, which returns a <code>Map</code> object of parameter names and values. Return different key-value pairs for different transformers associated with this entity by examining the <code>ITransformerInformation</code> object passed into the method.	Same as above.
Transformer Resource Bundle Constants used by all transformers for the content provider.	In the resource bundle located in the <code>/dist/PORTAL-INF/private/classes</code> folder of the PAR. The name of the resource bundle is defined in the <code><property></code> element named <code>ResourceBundleName</code> in the transformation service's <code><service-config></code> element.	From within the XSL transformer for this entity. For example: <pre><xsl:value-of select="resBundle.getString(\$ResourceBundle, 'STORY_TITLE')"/></pre>

3.1.3 Creating Administration Interfaces

The portal provides tools for building administration interfaces that enable portal administrators to create and edit portal content.

For example, you can build a portal component and create a wizard that guides administrators when they want to create a portal object based on the portal component. You can also create an editor that makes it easier for administrators to edit existing portal objects based on the component.

This section describes the following:

- [Creating Wizards \[Page 33\]](#)
- [Creating PCM Wizards \[Page 33\]](#)
- [Creating Editors \[Page 33\]](#)

3.1.3.1 Creating Wizards

This section describes how to use the wizard framework to create wizards for creating portal objects in the PCD.

Purpose

The wizard framework enables you to create administration wizards for creating new portal content. After creating and deploying a wizard, you can specify within a new portal component that your wizard be run whenever an administrator starts to create a new iView based on this component. edits an iView or other PCD object based on the new component.



The wizard framework enables you to create generic wizards. The wizard toolkit provides additional wizard functionality for creating iViews and other PCD objects.

For more information on the wizard toolkit, see the `com.sap.portal.admin.wizard` packages in the Javadocs.

This section describes the following:

- [Architecture \[Page 33\]](#): Background information on the wizard framework.
- [How to Create a Wizard \[Page 33\]](#): How to use the wizard framework API to build a wizard.
- [Essential Information \[Page 33\]](#): The `portalapp.xml` configuration and the JAR files for working with the wizard framework.

Assigning a Wizard

You can assign a wizard to a portal component so that when an administrator creates an iView or portal object from a template based on the portal component, the wizard is run.

To assign an editor to a portal component, set the `com.sap.portal.reserved.iview.WizardURL` property in the `<component-config>` element of the portal component's deployment descriptor to the full name of the wizard component. For more information on the wizard component, see [Step 2: Creating the Wizard Component \[Page 33\]](#).

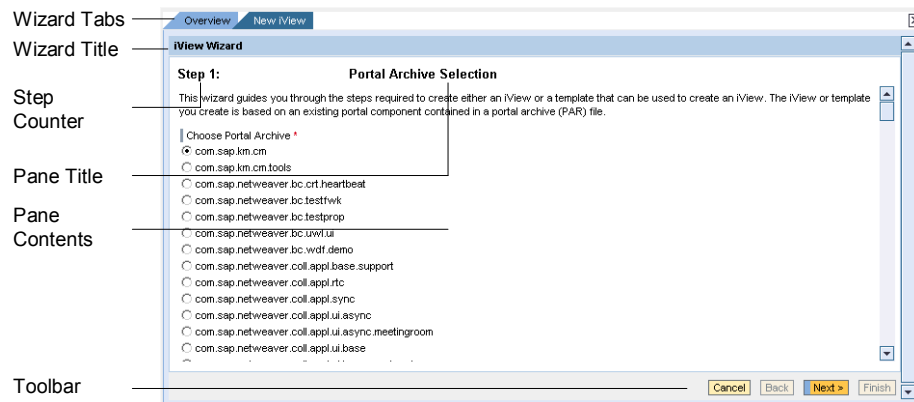
3.1.3.1.1 Architecture

This section provides background information on how the wizard framework works.

3.1.3.1.1.1 Wizard Look and Feel

Every wizard has a similar look-and-feel based on the wizard framework. The following is an example of a wizard, and how it is displayed in the portal:

Creating and Managing Content



Every wizard has the following parts:

- **Wizard Tabs:** One tab is displayed for each open wizard. The title depends on the purpose of the wizard, and is set by the component that launches the wizard.
- **Wizard and Pane Titles:** Set by the wizard component and each pane.
- **Step Counter:** The pane's position within the current wizard.
- **Pane Contents:** The user interface defined by the `AbstractWizardPane` class for the current pane.
- **Toolbar:** A set of buttons for navigating between panes in the wizard. The toolbar contains the following buttons:
 - **Cancel:** Close the wizard without taking any action.
 - **Back:** Go back to the previous pane.
 - **Next:** Go to the next pane.
 - **Finish:** Close the wizard and take any necessary action based on the input in the wizard. You must define the actions in a non-visible pane to which you navigate after *Finish* is clicked.

3.1.3.1.1.2 Wizard Framework Objects

A custom wizard consists of the following objects:

- **Wizard:** A portal component whose class extends `AbstractWizard` and which represents the overall wizard. The following are key wizard objects:
 - **Wizard Context** (`IWizardContext`): Holds the current state of the wizard, as well as the data model for holding data entered by the administrator when running the wizard.
 - **Configurable Wizard** (`IConfigurableWizard`): Used for configuring a wizard at startup, in `setupWizard()` of the `AbstractWizard` class.

The configurable wizard object can be used to create the following:

- **Transition:** Represents a navigation step between two panes in a wizard. A transition specifies a source and target pane, and the toolbar button that causes the navigation. You can also specify a condition, making the transition contingent on the condition being true.

Creating and Managing Content

You create transitions with the `IConfigurableWizard` object, but do not have access to transition objects.

- **Dependency:** Represents a relationship between two items in the data model, so that when the source item is changed the target item is erased.

You create dependencies with the `IConfigurableWizard` object, but do not have access to dependency objects.

For more information, see [Data Model \[Page 33\]](#).

- **Panes** (`AbstractWizardPane`): A set of Java classes, each of which implements `AbstractWizardPane` and represents one pane of the user interface.
- **Non-Visible Panes** (`AbstractActionState`): A set of Java classes, each of which implements `AbstractActionState`.

The non-visible pane is used to perform actions while navigating between two visible panes. It can also be used for the final pane, performing actions on the data provided by the administrator via the wizard.

- **Conditions** (`ICondition`): A class that implements the `ICondition` interface whose `isTrue()` method returns a `Boolean`. The class is used to add a condition to transitions.

For more information, see [Step 2: Creating the Wizard Component \[Page 33\]](#).

Key Classes/Interfaces

Class/Interface	Description
<code>AbstractWizard</code>	The wizard component extends this class.
<code>AbstractWizardPane</code>	Each wizard pane extends this class.
<code>AbstractActionState</code>	Each non-visible wizard pane extends this class.
<code>IWizardContext</code>	Represents the current state of the wizard
<code>IConfigurableWizard</code>	Provides methods for configuring the wizard.
<code>ICondition</code>	Each transition condition extends this class.
<code>IWizardComponent</code>	All panes and wizard UI components implement this interface.

3.1.3.1.1.3 Data Model

The data that is entered by an administrator in a wizard is generally kept in the wizard data model, a set of key-value pairs stored in the wizard context. The data model provides a way to keep wizard data so that it is accessible to all wizard panes.

Another key advantage of keeping the wizard's data in the data model is that the wizard framework can automatically store values entered in a UI control in the data model, and can automatically load into UI controls values set in the data model.

This synchronization – between data model and UI control – can only be done with wizard framework UI controls, which are part of the `com.sapportals.admin.wizardframework.components` package. These controls are based on HTMLB controls.

Creating and Managing Content

The wizard context is provided as an `IWizardContext` parameter in key methods of the wizard framework. For example, in `setupPane()` of a particular pane, the wizard context is supplied, as shown in the method's signature:

```
protected void setupPane(IWizardContext ctx, FlowContainer pane) {  
}
```

You can retrieve values from the data model and alter the user interface based on the values.

Wizard Framework Components

Wizard framework components are used to synchronize a data model item with a user interface control. The following are the available controls:

- `CheckboxChoiceComponent`
- `CheckboxSelectionComponent`
- `DropdownSelectionComponent`
- `Group`
- `HTMLScriptComponent`
- `ListDisplayComponent`
- `ListSelectionComponent`
- `MultilineInputComponent`
- `MultiSelectionComponent`
- `ProgressBarComponent`
- `RadioButtonSelectionComponent`
- `TextInputComponent`
- `TableViewComponent`

The components are all part of the `com.sapportals.admin.wizardframework.components` package.

To synchronize a UI control with a specific item in the data model, call `setValueTargetpath()` on the control and pass the name of the data model item that will store the control's value.

For example, the following creates an input field called `Last Name`, whose value is associated with the data model item `details.lastName`.

```
TextInputComponent firstName = new TextInputComponent("Last Name");  
firstName.setValueTargetPath("details.lastName");
```

Dependencies

You may want to erase specific data model items when the value for another item changes. You can set up this dependency by calling `addDependency()` on the `IConfigurableWizard` object and supplying the source and target data model items. This is done in `setupWizard()` of the `AbstractWizard` object.

The following causes all items with the string prefix of the value in `AddressPane.ADDRESS_PANE_KEY` to be erased whenever the `last_name` item is changed:

```
wizard.addDependency("last_name", AddressPane.ADDRESS_PANE_KEY);
```

3.1.3.1.1.4 Process Flow

The following is the wizard framework process flow, showing the methods that are called on the `AbstractWizardPane` object and the user events that occur, starting from when a pane is first displayed:

- **setupPane()**: Creates the user interface for the pane.
- **doBeforeDisplay()**: Performs synchronization between the data model and the pane. The default implementation calls the pane's `myDoBeforeDisplay()` method, and then calls `doBeforeDisplay()` on all the wizard components within the pane.

Implement `myDoBeforeDisplay()` to set the values of HTMLB controls in your pane based on the data model.
- **User Input**: The user enters data into the user interface controls, and then clicks a toolbar button.
- **processInput()**: Performs synchronization between the pane and the data model, saving to the data model the values entered into wizard controls by the administrator. The default implementation calls the pane's `myProcessInput()` method, and then calls `processInput()` on all the wizard components within the pane.

Implement `myProcessInput()` to save to the data model the data from any HTMLB controls that you created in the pane.

The `processInput()` method is called after a user clicks a toolbar button, causing a new request to be made to the wizard component.
- **getErrorMessages()**: Validates the data for the pane. If the data is valid, the method returns `null`. If the data is not valid, the method returns a `List` of error messages, and flow returns to `setupPane()`, above.
- **doAfterSubmit()**: Performs any actions on the data if the data is valid.

The wizard then navigates to the next pane based on the transitions set by the wizard, and the flow starts again with `setupPane()`, above.

Non-visible Panes

For a non-visible pane, the wizard framework simply calls `doAction()` on the `AbstractActionState` object, and then navigates to the next pane based on the transitions set by the wizard. The flow starts again with `setupPane()`, above.

3.1.3.1.2 How to Create a Wizard

A wizard is a portal application, deployed in a PAR, with the following parts:

- **Wizard Component**: A portal component that represents the overall wizard.
- **Panes**: Each pane is a Java class that defines the user interface for one section of the wizard.
- **Conditions**: A Java class that defines a condition for transitioning from one pane to another.

Workflow

The following are the steps required for creating a wizard:

1. [Step 1: Creating Panes \[Page 33\]](#)

Creating and Managing Content

2. [Step 2: Creating the Wizard Component \[Page 33\]](#)
3. [Step 3: Creating Conditions \[Page 33\]](#)

3.1.3.1.2.1 Step 1: Creating Panes

A wizard contains one or more panes that contain the input fields, buttons and other controls that make up the user interface.

Procedure

For each pane, perform the following:

1. Create a new class that extends `AbstractWizardPane`.
2. Implement `setupPane()`, in which you build a user interface with the help of HTMLB and wizard framework components. The following is the method's signature:

```
protected void setupPane(IWizardContext ctx, FlowContainer pane) {
}
```

3. Create your user interface with HTMLB and wizard framework controls that you add to the `FlowContainer` object passed into the method.

For example, the following adds an input field called `first_name`, whose value is associated with the data model field `info.firstName`.

```
TextInputComponent firstNameComp =
    new TextInputComponent("first_name", "");
firstNameComp.setValueTargetPath("info.firstName");
pane.addComponent("first_name", firstNameComp, false);
```

It is best to use wizard framework controls, as these can be synchronized with the data model.

For more information on wizard framework controls, see [Data Model \[Page 33\]](#).

For more information on HTMLB controls, see [HTML-Business for Java \[Page 33\]](#).

Options

The following lists additional methods that you can implement:

- `getTitle()`: Sets the pane title.
- `myDoBeforeDisplay()`: Performs any initialization before the pane is displayed, such as setting the values of HTMLB controls from the data model.
- `myProcessInput()`: Processes data entered into the wizard UI, and saves to the data model the data from any HTMLB controls in the pane.

3.1.3.1.2.2 Step 2: Creating the Wizard Component

Each wizard contains a wizard component, whose Java class extends `AbstractWizard`. This class is descended from `AbstractPortalComponent`, making the wizard a standard `iView` that can be displayed in the portal administration pages.

`AbstractWizard` requires you to implement one method, `setupWizard()`, which is passed an `IConfigurableWizard` object. In this method, the wizard does the following:

- Sets the wizard title.

Creating and Managing Content

- Adds panes to the `IConfigurableWizard` object
- Sets the transitions between panes.
- Sets dependencies between data model objects.

Procedure

1. Create a new class that extends `AbstractWizard`.
2. Implement `setupWizard()`, which has the following signature:

```
public void setupWizard(
    com.sapportals.admin.wizardframework.api.IConfigurableWizard
    wizard,
    IPortalComponentProfile profile,
    java.util.Map params,
    java.util.ResourceBundle bundle) {
}
```

The method receives an `IConfigurableWizard` object, which is used to create panes, add transitions, and add dependencies.

In this method, do the following:

- **Create Panes:** To create and add a pane to the wizard, supply a key for the pane and the pane's class name, as shown in the following example:

```
wizard.addPane(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
    PersonalDetailsPane.class.getName());
```

- **Add Transitions:** Transitions determine to what panes the wizard navigates when each button is clicked.



You only need transitions if the wizard is not a linear set of panes. If no transitions are created, the wizard navigates from pane to pane in the order they were added to the wizard.

The following enables the *Next* button when the address pane is displayed, and specifies that the wizard display the table pane when the button is clicked:

```
wizard.addTransition(AddressPane.ADDRESS_PANE_KEY,
    NEXT, null, TablePane.TV_PANE_KEY);
```

Each transition can also contain a condition that must return `true` in order for the transition to take effect. If the condition is `false`, the navigation specified by the transition does not take place.

A condition is an instance of a class that extends `ICondition`. For more information, see [Step 3: Creating Conditions \[Page 33\]](#).

The following is an example of a transition with a condition:

```
wizard.addTransition(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
    NEXT, stuCond, StudentPane.STUDENT_PANE_KEY);
```

- **Add Dependencies:** Dependencies cause objects in the data model to be deleted whenever the value of other objects change. For example, you may want to clear the address field if the administrator changes the name field.

The following clears all properties that start with the address pane key if the last name in the personal details pane was changed:

Creating and Managing Content

```
wizard.addDependency(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY  
+ ".last_name", AddressPane.ADDRESS_PANE_KEY);
```

3. Create a `<component>` element in the `portalapp.xml` for the wizard component, similar to the element for a standard component.

Add a property element called `isStateless` in the `<component-profile>` element called to determine if session data is maintained on the client (`true`) or on the server (`false`).

```
<components>  
  <component name="trainSample">  
    <component-config>  
      <property name="ClassName" value="TrainWizard"/>  
      <property name="LocalModeAllowed" value="true"/>  
      <property name="SafetyLevel" value="low_safety"/>  
    </component-config>  
    <component-profile>  
      <property name="isStateless" value="true"/>  
    </component-profile>  
  </component>  
</components>
```

Result

The following is an example of `setupWizard()` that creates:

- 6 panes
- A close pane that closes the wizard
- A condition (`isStudent`)
- Several transitions that define how the wizard navigates between panes
- One data dependency

Creating and Managing Content

```

public void setupWizard(
    com.sapportals.admin.wizardframework.api.IConfigurableWizard
wizard,
    IPortalComponentProfile profile,
    java.util.Map params,
    java.util.ResourceBundle bundle) {

    wizard.setTitle("Order Train Tickets");

    wizard.addPane(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
        PersonalDetailsPane.class.getName());
    wizard.addPane(AddressPane.ADDRESS_PANE_KEY,
        AddressPane.class.getName());
    wizard.addPane(StudentPane.STUDENT_PANE_KEY,
        StudentPane.class.getName());

    wizard.addPane("close", com.sapportals.admin.wizardframework.
        components.CloseWizardComponent.class.getName());
    wizard.addPane(TicketInfo.TICKET_INFO_PANE_KEY,
        TicketInfo.class.getName());
    wizard.addPane>ShowPanes.SHOW_PANE_KEY,
        ShowPanes.class.getName());
    wizard.addPane(TablePane.TV_PANE_KEY, TablePane.class.getName());

    //creating an instance of the ICondition, IsStudent
    IsStudent stuCond = new IsStudent();

    //creating the reverse condition from IsStudent
    ICondition notStuCond = new IsStudent() {
        public boolean isTrue(IWizardContext ctx) {
            return !super.isTrue(ctx);
        }
    };

    wizard.addTransition(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
        NEXT, stuCond, StudentPane.STUDENT_PANE_KEY);

    wizard.addTransition(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
        NEXT, notStuCond, AddressPane.ADDRESS_PANE_KEY);

    wizard.addTransition(AddressPane.ADDRESS_PANE_KEY, BACK, stuCond,
        StudentPane.STUDENT_PANE_KEY);
    wizard.addTransition(AddressPane.ADDRESS_PANE_KEY, BACK,
        notStuCond, PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY);

    wizard.addTransition(StudentPane.STUDENT_PANE_KEY, BACK, null,
        PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY);
    wizard.addTransition(StudentPane.STUDENT_PANE_KEY, NEXT, null,
        AddressPane.ADDRESS_PANE_KEY);

    wizard.addTransition(AddressPane.ADDRESS_PANE_KEY, NEXT, null,
        TablePane.TV_PANE_KEY);
    wizard.addTransition(TicketInfo.TICKET_INFO_PANE_KEY, BACK, null,
        AddressPane.ADDRESS_PANE_KEY);

    wizard.addTransition(TicketInfo.TICKET_INFO_PANE_KEY, NEXT, null,
        TablePane.TV_PANE_KEY);

```

Creating and Managing Content

```

wizard.addTransition(TablePane.TV_PANE_KEY, NEXT, null,
    ShowPanels.SHOW_PANE_KEY);
wizard.addTransition(TablePane.TV_PANE_KEY, BACK, null,
    AddressPane.ADDRESS_PANE_KEY);

wizard.addTransition(ShowPanels.SHOW_PANE_KEY, FINISH, null,
    "close");
wizard.addTransition(ShowPanels.SHOW_PANE_KEY, BACK, null,
    TablePane.TV_PANE_KEY);

wizard.addDependency(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY
    + ".last_name", AddressPane.ADDRESS_PANE_KEY);
}

```

3.1.3.1.2.3 Step 3: Creating Conditions

Conditions can be added to transitions to determine whether the navigation defined in the transition occurs.

In a transition, you specify a button, a condition, a source pane and a target pane. If the user clicks the button in the source pane, the wizard navigates to the target pane only if the condition is true. If no condition is specified (that is, `null`), the transition always occurs if the button is clicked.

For more information on transitions, see [Step 2: Creating the Wizard Component \[Page 33\]](#).

Procedure

1. Create a new class that implements `ICondition`.
2. Implement `isTrue()`, which has the following signature:

```

public boolean isTrue(IWizardContext ctx) {
}

```

Using Conditions

The following describes how to use the `ICondition` class:

1. When creating transitions in your wizard component, create an instance of the condition class, as in the following example:

```

IsStudent stuCond = new IsStudent();

```

2. If needed, create another condition class that checks if the original condition is false, as in the following example:

```

ICondition notStuCond = new IsStudent() {
    public boolean isTrue(IWizardContext ctx) {
        return !super.isTrue(ctx);
    }
};

```

3. Specify the condition in a transition. In the following example, the *Next* button on the personal details pane navigates to the student pane if the user is a student, and to the address pane if the user is not a student:

```
wizard.addTransition(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
    NEXT, stuCond, StudentPane.STUDENT_PANE_KEY);

wizard.addTransition(PersonalDetailsPane.PERSONAL_DETAILS_PANE_KEY,
    NEXT, notStuCond, AddressPane.ADDRESS_PANE_KEY);
```

3.1.3.1.3 Essential Information

This section provides the following information to help you create wizards:

- [portalapp.xml \[Page 33\]](#): Information about the `portalapp.xml` file for a wizard application.
- [JARs and Packages \[Page 33\]](#): The packages and JAR files required for creating a wizard.

3.1.3.1.3.1 portalapp.xml

The following is a sample `portalapp.xml` for a wizard:

```
<application>
  <application-config>
    <property name="SharingReference"
      value="com.sap.portal.admin.wizardframework"/>
    <property name="Vendor" value="sap.com"/>
    <property name="SecurityArea" value="NetWeaver.Portal"/>
  </application-config>
  <services>
  </services>
  <components>
    <component name="trainSample">
      <component-config>
        <property name="ClassName" value="TrainWizard"/>
        <property name="LocalModeAllowed" value="true"/>
        <property name="SafetyLevel" value="low_safety"/>
      </component-config>
      <component-profile>
        <property name="isStateless" value="true">
        </property>
      </component-profile>
    </component>
  </components>
</application>
```

Application Configuration Element

In the `<application-config>` element, add a `SharingReference` for the following applications:

Service/Component	SharingReference
Wizard Framework	<code>com.sap.portal.admin.wizardframework</code>

```
<application-config>
  <property name="releasable" value="false"/>
```

```
<property name="Vendor" value="sap.com"/>
<property name="SecurityArea" value="NetWeaver.Portal"/>
<property name="SharingReference"
  value="com.sap.portal.admin.wizardframework" />
</application-config>
```

Components Element

A wizard contains one portal component, the wizard component, which is based on a class that extends `AbstractWizard`. For more information on creating the editor component's `<component>` element, see [Step 2: Creating the Wizard Component \[Page 33\]](#).

```
<components>
  <component name="trainSample">
    <component-config>
      <property name="ClassName" value="TrainWizard"/>
      <property name="LocalModeAllowed" value="true"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name="isStateless" value="true"/>
    </component-profile>
  </component>
</components>
```

3.1.3.1.3.2 JARs and Packages

Packages

- `com.sapportals.admin.wizardframework`

JARs

- `com.sap.portal.admin.wizardframework_api.jar`

3.1.3.2 Creating PCM Wizards

This section describes how to use the wizard toolkit to create a PCM wizard for administrators.

Purpose

The wizard toolkit enables you to create administration wizards for creating new portal objects, which are stored in the PCD. The wizard toolkit is built on top of the wizard framework, and adds the following functionality:

- Automatically synchronizes wizard input fields with the properties of the portal object to be created.
- Adds a set of standard panes that are required for creating all types of portal objects.

This section describes the following:

Creating and Managing Content

- [Architecture \[Page 33\]](#): Provides background information on the wizard toolkit.
- [How to Create a PCM Wizard \[Page 33\]](#): Describes how to use the wizard toolkit API to build a PCM wizard.
- [Essential Information \[Page 33\]](#): Describes the `portalapp.xml` configuration and the JAR files for working with the wizard toolkit.

Prerequisites

- Familiarity with the wizard framework and how to create standard wizards. For more information, see [Creating a Wizard \[Page 33\]](#).

3.1.3.2.1 Architecture

This section provides background information on how the wizard toolkit works.



A PCM wizard is only launched when you create a new iView from an iView template, and not when creating an iView from a PAR.

3.1.3.2.1.1 PCM Wizard Look and Feel

PCM wizards are similar to standard wizards. All wizards contain a set of panes and a toolbar at the bottom of the wizard for navigating between panes.

PCM wizards also come with the following predefined panes that are displayed or activated automatically:

- **Init** (non-visible): Initializes the wizard, such as setting the title and other parameters.
- **Info**: Provides a user interface to enable the administrator to enter standard fields for all objects, such as the name and ID for the object.

- **Summary**: Displays a summary of the information entered by the administrator, and enables the administrator to create the object or go back and change the information.

Creating and Managing Content

iView Wizard	
Step 4:	Summary
Based on:	par:/applications/com.sap.portal.admin.pcmobjectwizardtoolkit/components/MailViewWizard
Computer Number:	Number
Email Address:	Address
iView ID:	myObject
iView Name:	myObject
Master Language:	English
Open Mail:	yes
Remember Password:	yes
Server Type:	Pop3 e-Mail Server
User Name:	User
Your Name:	Name
<div> <input type="button" value="Cancel"/> <input type="button" value="Back"/> <input type="button" value="Next >"/> <input type="button" value="Finish"/> </div>	

- **Save (non-visible):** Saves the new object with all the properties defined by the wizard.
- **FinishPane:** Displays after the new object is created, and provides options to:
 - Open the editor for the new object.
 - Rerun the wizard to create another object.
 - Close the wizard.

Choose your next step:

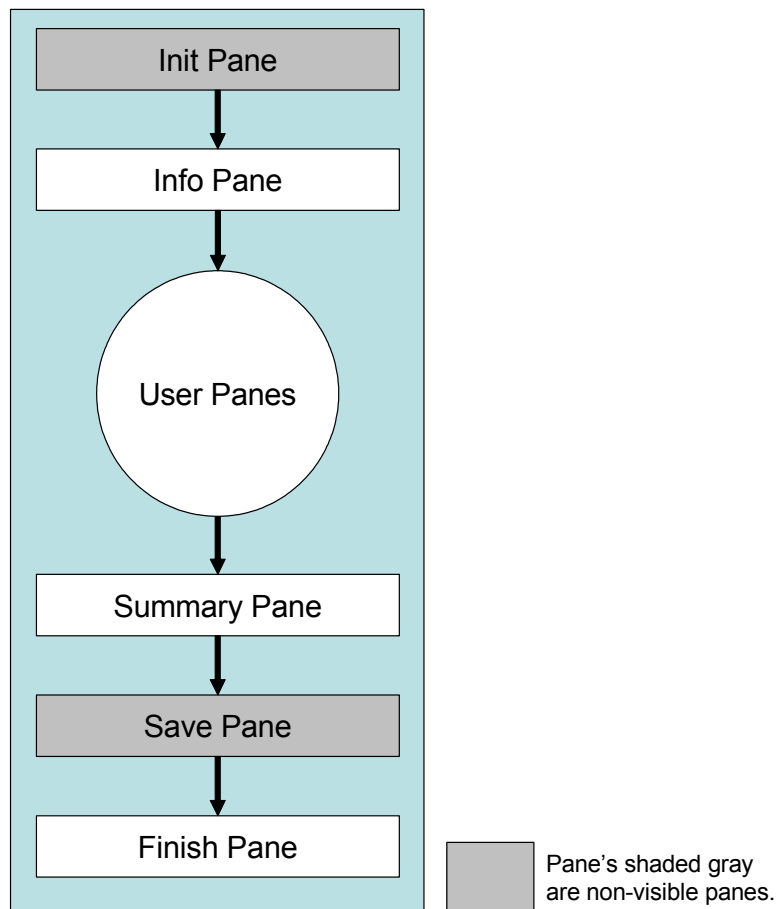
- ☒ Open the object for editing
- ☐ Restart the wizard to create a new object using the same portal component
- ☐ Close the wizard

For your custom functionality, you create custom panes that are displayed after the Init pane and before the Summary pane.

The process flow, that is, when each pane is displayed or called, is described in [Process Flow \[Page 33\]](#).

3.1.3.2.1.2 Process Flow

The following shows the order in which the wizard toolkit activates the panes in a PCM wizard:



User panes are panes created by you specifically for your wizard. All other panes are automatically created and called by the wizard toolkit.

3.1.3.2.1.3 Wizard Toolkit Objects

PCM wizards contain the same objects as standard wizards, such as wizards, panes, transitions and conditions, with the following changes:

- The wizard component extends `AbstractPCMWizard` (not `AbstractWizard`), which automatically provides the default panes and transitions for PCM wizards.
- The wizard toolkit contains helper classes for creating input controls and synchronizing them with properties of portal objects.

Key Classes/Interfaces

Shaded objects are new to the wizard toolkit. All other objects are inherited from the wizard framework.

Class/Interface	Description
<code>AbstractPCMWizard</code>	The PCM wizard component extends this class.
<code>AbstractWizardPane</code>	Each wizard pane extends this class.
<code>AbstractActionState</code>	Each non-visible wizard pane extends this class.

Creating and Managing Content

IWizardContext	Represents the current state of the wizard.
IConfigurableWizard	Provides methods for configuring the wizard.
ICondition	Each transition condition extends this class.
IWizardComponent	All panes and wizard UI components implement this interface.
IBasicObjectCreationService	A helper class for working with PCM objects. For more information, see Synchronizing with the Property Editor [Page 33] .
IWizardComponentFactory	A factory class for creating the appropriate wizard control for a specific object property. For more information, see Synchronizing with the Property Editor [Page 33] .

3.1.3.2.1.4 Synchronizing with the Property Editor

One of the key features of the wizard toolkit is that you can synchronize wizard controls with properties of the object to be created. The wizard can automatically save the information entered into a control by the administrator to the corresponding property in the object.

When creating the user interface for a pane in the pane's `setupPane()` method, you create wizard controls and synchronize them to a property of the object to be created.

For example, a portal component defines a property called `Email Address`, and a template object based on this component exists in the portal. In the wizard for the component, you can create a pane that includes a wizard component for this property, as follows:

1. Get the ID of the template by using an `IBasicObjectCreationService` helper object.

Use the standard `PortalRuntime.getRuntimeResources().getService()` method for creating this helper object.

```
IBasicObjectCreationService basicObjService =
    (IBasicObjectCreationService) PortalRuntime.getRuntimeResources()
        .getService(IBasicObjectCreationService.KEY);

String targetId = _basicObjService.getTargetIdParam(context);
```

2. Create the appropriate control for the `Email Address` property by using an `IWizardComponentFactory` object. The `IWizardComponentFactory` object queries the template to determine the appropriate control.

Use the standard `PortalRuntime.getRuntimeResources().getService()` method for creating this factory object.

```
IWizardComponentFactory wizardCompFactory =
    (IWizardComponentFactory) PortalRuntime.getRuntimeResources()
        .getService(IWizardComponentFactory.KEY);

AbstractInputComponent emailAddress =
    wizardCompFactory.getControlForAttribute(
        targetId, "Email Address", context.getRequest());
```

3. Associate the control to the `Email Address` property of the object to be created.

```
_basicObjService.setComponentAsAttribute(  
    emailAddress, "Email Address", true, context);
```

You can then add the component to the pane by calling the pane's `addComponent()` method.

3.1.3.2.1.5 Data Model

The data model for PCM wizards is the same as for standard wizards. You can store key-value pairs in the data model, whose data you can access via an `IWizardContext` object.

Individual key-value pairs can be automatically associated with specific wizard controls via the control's `setValueTargetPath()` method.

However, the wizard toolkit enables you to directly synchronize controls to properties of the portal object to be created, without storing the control's value to the data model.

For more information about the data model, see [Data Model \[Page 33\]](#).

3.1.3.2.2 How to Create a PCM Wizard

A PCM wizard is a portal application, deployed in a PAR, with the following parts:

- **PCM Wizard Component:** A portal component that represents the overall PCM wizard.
- **Panes:** Each user pane is a Java class that defines the user interface for one section of the wizard.
- **Conditions:** A Java class that defines a condition for transitioning from one pane to another.

Workflow

The following are the steps required for creating a PCM wizard:

1. [Step 1: Creating User Panes \[Page 33\]](#)
2. [Step 2: Creating a PCM Wizard Component \[Page 33\]](#)
3. [Step 3: Creating Conditions \[Page 33\]](#)

3.1.3.2.2.1 Step 1: Creating User Panes

A PCM wizard automatically contains several standard panes, such as the Info and Finish panes. This procedure describes how to create your own panes for collecting from the administrator additional data specific to your portal component.

Procedure

For each custom pane, follow the procedure for creating panes for a standard wizard, as described in [Step 1: Creating Panes \[Page 33\]](#).

For PCM wizard panes, synchronize your input controls with the properties in the portal object to be created, as described in [Synchronizing with the Property Editor \[Page 33\]](#).

3.1.3.2.2 Step 2: Creating a PCM Wizard Component

Each PCM wizard contains a PCM wizard component, whose Java class extends `AbstractPCMWizard`. This class is descended from `AbstractPortalComponent`, making the wizard a standard `iView` that can be displayed in the portal administration pages.

`AbstractPCMWizard` enables you to implement several methods for handling the following tasks:

- Adding user panes, which are in addition to the ones automatically added to the wizard by the `AbstractPCMWizard` class.
- Setting transitions between panes.
- Setting dependencies between data model objects. This is not so important because data entered into the wizard by an administrator can be automatically synchronized to the properties of the portal object to be created, and does not require use of the wizard framework data model
- Modifying the text that is displayed in the Summary pane.
- Modifying the property values that are saved to the portal object to be created. This allows you to make last-minute changes to the values entered by the administrator.

Procedure

1. Create a new class that extends `AbstractPCMWizard`.
2. Implement `addUserPanes()`, which has the following signature:

```
protected void addUserPanes(IConfigurableWizard wizard) {
}
```

Add the panes for the wizard by calling `addPane()` on the `IConfigurableWizard` object passed into the method, as follows:

```
wizard.addPane(ServerInfo.SERVER_INFO_PANE_KEY,
    ServerInfo.class.getName());
```

The above assumes that `ServerInfo` is the name of a class that defines a pane, as described in [Step 1: Creating User Panes \[Page 33\]](#).

3. Implement `addTransitions()`, which has the following signature:

```
protected void addUserTransitions(IConfigurableWizard wizard) {
}
```

Add transitions between the wizard panes. For more information on transitions, see [Wizard Framework Objects \[Page 33\]](#).

The wizard automatically starts the Info pane. You must supply transitions from this pane. The following example creates a wizard that displays the built-in Info pane, the custom `UserInfo` pane, the custom `ServerInfo` pane, and then the built-in Summary and Finish panes, in that order:

```
//Built-in Info pane to custom UserInfo pane
wizard.addTransition(START_PANE, NEXT, null,
    UserInfo.USER_INFO_PANE_KEY);

//UserInfo pane to custom ServerInfo pane
wizard.addTransition(UserInfo.USER_INFO_PANE_KEY, NEXT, null,
    ServerInfo.SERVER_INFO_PANE_KEY);

//ServerInfo pane to built-in Summary and Finish panes
```

Creating and Managing Content

```
wizard.addTransition(ServerInfo.SERVER_INFO_PANE_KEY, NEXT, null,
    END_PANE);
```

4. Implement `finalizeSummaries()`, in order to modify what is displayed on the summary pane, if necessary.

The method is passed a `Map` object with key-value pairs that are displayed in the summary pane. You can modify existing key-value pairs, or add additional pairs.

The following sample takes a key-value pair from the data model (`Open Mail`) and adds it to the summary `Map`, and then adds a prefix to an existing key-value pair (`path`) in the summary `Map`:

```
public void finalizeSummaries(Map summaries, IWizardContext context)
{
    String openMail = (String) context.getProperty("openMail");
    summaries.put("Open Mail", openMail);

    String path = (String) summaries.get("path");
    path = "http://" + path;
    attributes.put("path", path);
}
```

5. Implement `finalizeDescriptor()`, in order to modify the property values that are stored for the object that you are creating with the wizard.

The method is passed a `Map` object with key-value pairs that are the property values to be stored for the new object. Add or modify values, as needed.

At the end of the method, call `super.finalizeDescriptor()`.

6. Create a `<component>` element in the `portalapp.xml` for the PCM wizard component, similar to the element for a standard component and a standard wizard.

Add a property element called `isStateless` in the component-profile element to determine if session data is maintained on the client (`true`) or on the server (`false`).

```
<components>
  <component name="pcmSample">
    <component-config>
      <property name="ClassName" value="pcmWizard"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name="isStateless" value="true"/>
    </component-profile>
  </component>
</components>
```

Options

In addition to the above steps, you may also want to do the following:

- **Set UI Strings:** Modify the user-interface strings displayed on the panes automatically created by the wizard toolkit, that is, the Info, Summary and Finish panes.

To modify the strings, implement the `setUIStrings()` method, which takes a `Map` object as a parameter. To modify a string, add a key-value pair, indicating the string to modify and the new string.

The key is a constant in the class of one of the pre-defined wizard toolkit panes.

Creating and Managing Content

The value is a key in a resource bundle that you attach to your PCM wizard component. You can attach a resource bundle to any portal component, as described in [Internationalization \[Page 27\]](#).

For example, to change the question displayed on the Finish pane, add to the `Map` the following:

- **Key:**
FinishPane.UIStrings.POST_COMPLETION_CHOICE_CAPTION_BUNDLE_KEY
- **Value:** The resource bundle key of the new string

```
protected void setUIStrings(Map map) {
    map.put(FinishPane.UIStrings.POST_COMPLETION_CHOICE_CAPTION_BUNDLE_KEY,
        "NEWSTRING");
}
```

- **Initialize Wizard for Testing:** When the wizard is launched, the wizard toolkit automatically sets initial parameters, such as the location of the new portal object or the portal component on which to base the new portal object.

During testing, you may run the PCM wizard component as standalone, and now through the administration tools. If so, you need to set some initial parameters, as shown in the following example:

```
public void initWizardSession(IWizardContext context) {
    context.getWizardParameters().put(
        ObjectCreationWizardConstants.SAVE_LOCATION_PARAM,
        "pcd:portal_content/TestObject");
    context.getWizardParameters().put(
        ObjectCreationWizardConstants.TARGET_ID_PARAM,
        "pcd:portal_content/DanielContent/myHelloTemplate");
    context.getWizardParameters().put(
        ObjectCreationWizardConstants.CREATE_MODE_PARAM,
        "com.sapportals.portal.application.Component");
    context.getWizardParameters().put(
        ObjectCreationWizardConstants.OBJECT_TYPE_PARAM,
        "com.sapportals.portal.iview");
    super.initWizardSession(context);
}
```

3.1.3.2.3 Step 3: Creating Conditions

Conditions are created in the same way as conditions for standard wizards.

For more information, see [Step 3: Creating Conditions \[Page 33\]](#).

3.1.3.2.3 Essential Information

This section provides the following information to help you create PCM wizards:

- [portalapp.xml \[Page 33\]](#): Information about the `portalapp.xml` file for a PCM wizard application.
- [JARs and Packages \[Page 33\]](#): The packages and JAR files required for creating a PCM wizard.

3.1.3.2.3.1 portalapp.xml

The following is a sample `portalapp.xml` for a PCM wizard:

Creating and Managing Content

```

<application>
  <application-config>
    <property name="SharingReference"
      value="com.sap.portal.admin.wizardframework"/>
    <property name="Vendor" value="sap.com"/>
    <property name="SecurityArea" value="NetWeaver.Portal"/>
  </application-config>
</services>
</services>
<components>
  <component name="trainSample">
    <component-config>
      <property name="ClassName" value="pcmWizard"/>
      <property name="LocalModeAllowed" value="true"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name="isStateless" value="true">
    </property>
    </component-profile>
  </component>
</components>
</application>

```

Application Configuration Element

In the `<application-config>` element, add a `SharingReference` for the following services:

Service/Component	SharingReference
Wizard Toolkit	com.sap.portal.admin.pcmobjectwizardtoolkit

```

<application-config>
  <property name="SharingReference"
    value="com.sap.portal.admin.pcmobjectwizardtoolkit" />
</application-config>

```

Components Element

A PCM wizard contains one portal component, the PCM wizard component, which is based on a class that extends `AbstractPCMWizard`. For more information on creating the editor component's `<component>` element, see [Step 2: Creating a PCM Wizard Component \[Page 33\]](#).

```

<components>
  <component name="pcmSample">
    <component-config>
      <property name="ClassName" value="pcmWizard"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name="isStateless" value="true">
    </property>
    </component-profile>
  </component>
</components>

```

3.1.3.2.3.2 JARs and Packages

Packages

- `com.sapportals.admin.pcmobjectwizardtoolkit`

JARs

- `com.sap.portal.admin.pcmobjectwizardtoolkit_api.jar`

3.1.3.3 Creating Editors

This section describes how to use the editor framework to create editors for editing portal objects in the PCD.

Purpose

The editor framework enables you to create administration editors for configuring iViews and other PCD objects. These editors are displayed when a portal administrator edits an existing PCD object.

Once deployed, an editor can be assigned to a specific PCD object, so that when that object is edited, the corresponding editor is displayed. More commonly, an editor is assigned to a portal component in a PAR file, so that when an administrator edits any iView based on that component, the corresponding editor is displayed.

This section describes the following:

- [Architecture \[Page 33\]](#): Background information on the editor framework.
- [How to Create an Editor \[Page 33\]](#): How to use the editor framework API to build an editor.
- [Essential Information \[Page 33\]](#): The `portalapp.xml` configuration and the JAR files for creating an editor.

Assigning an Editor

You can assign an editor to a portal component so that when an administrator edits an iView or portal object based on the portal component, the editor is run.

To assign an editor to a portal component, set the

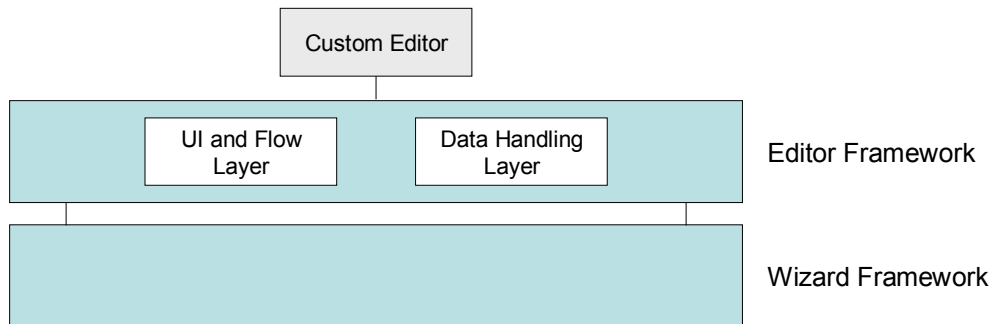
`com.sap.portal.reserved.iview.EditorURL` property in the `<component-config>` element of the portal component's deployment descriptor to the full name of the editor component. For more information on the editor component, see [Step 2: Creating an Editor Component \[Page 33\]](#).

Prerequisites

You should be familiar with the Portal Content Studio, which provides a central environment for content administrators to develop and manage portal content. For more information on the Portal Content Studio, see [Portal Content Studio \[External\]](#).

3.1.3.3.1 Architecture

The editor framework architecture is composed of the following parts:



- **Wizard Framework:** A set of APIs on which the editor framework is built, and provides such things as a set of components for use in editors and a data model for maintaining edited data.
- **Editor Framework:** A set of APIs that are divided into the following parts:
 - **UI and Flow Layer:** Builds and renders the editor, manages the data that is edited in the editor and operates the data handling layer.
 - **Data Handling Layer:** Loads and saves the data that is edited in the editor.
- **Custom Editor:** A portal application that contains Java classes and other files for implementing a custom editor for editing PCD objects.

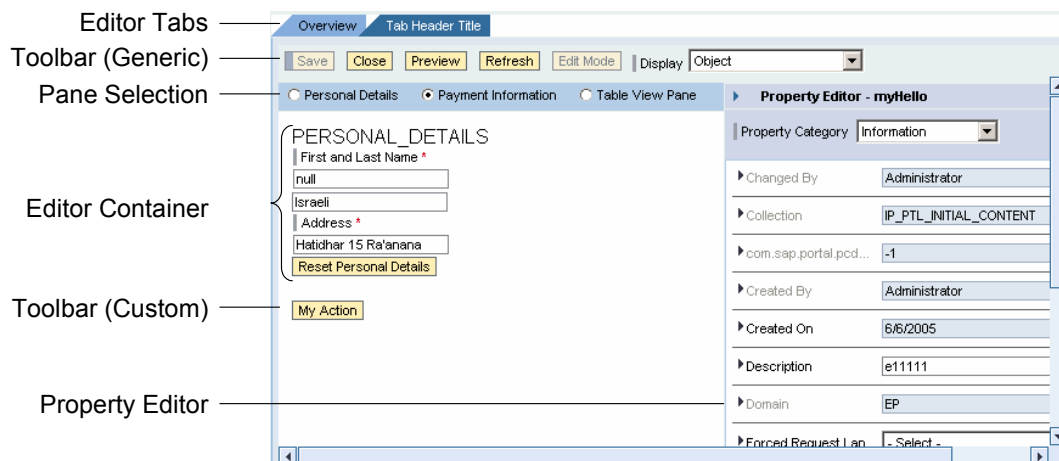
For information on building a custom editor, see *How to Build an Editor*.

3.1.3.3.1.1 Editor Look and Feel

An editor defines a set of panes, each of which contains a portion of the user interface for editing the portal object. The panes are displayed within the editor framework, which provides some standard editor functionality and a standard look and feel for every editor.

The following shows the basic parts of an editor:

Creating and Managing Content



Every editor has the following parts:

- **Editor Tabs:** One tab is displayed for each open editor. The editor can set the title displayed in its tab. The default title is the name of the PCD object being edited.
- **Toolbars:** A collection of buttons and controls for working with the editor. The following toolbars are available:
 - **Generic:** This toolbar is displayed at the top of the editor and contains a fixed set of standard buttons, such as *Save* and *Close*. This toolbar is displayed for all panes and cannot be modified, though some of the functionality triggered by the toolbar's controls can be altered.
 - **Custom:** On each pane, you can create your own toolbar of buttons, and then capture the events triggered by the buttons on the server side.
- **Pane Selection:** A radio button group that enables the administrator to select a pane to be displayed.
- **Editor Container:** The area in each pane where your custom user interface is displayed.
- **Property Editor:** A built-in user interface that displays the properties of a PCD object, and enables an administrator to edit them (if they are not read only).

3.1.3.3.1.2 Editor Framework Objects

A custom editor makes use of the following objects:

- **Editor:** A portal component whose class extends `AbstractEditorComponent` and which represents the overall editor. The following are key editor objects:
 - **Editor Context** (`IEditorContext`): Holds the current state of editor, as well as the data model that holds the data currently being edited. The editor context is an `IEditorContext` object, which is generally supplied as a parameter in the key methods of the editor framework.
 - **Editor Response** (`IEditorResponse`): Provides access to the portal response object, and enables you to do the following:
 - **Set Save Question:** The save question is displayed whenever the administrator clicks *Save* in the generic toolbar.

Creating and Managing Content

- **Set Administrator Message:** This message is displayed in the standard portal message bar at the end of the current request.

You may want to display a message, for example, after you throw an exception to cancel the saving of the editor's data, or after displaying the user interface for an editor pane.

- **Set Client Properties:** These properties are set on the server side but are available on the client side in `parametersMap`, a JavaScript object. Values are obtained via the `get()` method.

The editor response is an `IEditorResponse` object, and is available from the editor context.

- **Panes:** A set of Java classes, each of which implements `IEditorPane` and represents one pane of the user interface. Each pane can have the following characteristics:
 - **Visible/Hidden:** If a pane is visible, a radio button is displayed for it in the pane selection area at the top of the editor. If a pane is invisible, the pane can only be displayed via code, through an activating an event.
 - **Default Pane:** The default pane is the pane displayed when the editor is launched.
 - **Activating Event:** A pane's activating event is the server event that causes the editor to navigate to that pane.

A pane's properties are set when it is created in the editor component's `setupEditor()` method.

- **Data Handler:** A Java class that extends `PCMDDataHandler` that defines methods for loading data into the editor context and saving data from the editor context.

To make the data handler accessible to editors defined in other PARs, the data handler must be defined as a service and implement `IService`.

For more information on the editor context and data model, see [Data Model \[Page 33\]](#).

Key Classes/Interfaces

Class/Interface	Description
<code>AbstractEditorComponent</code>	Extended by the editor component.
<code>IEditorContext</code>	Represents the current state of the editor.
<code>IEditorResponse</code>	Provides access to the portal response, as well as additional functionality for the editor framework.
<code>IEditorPane</code>	Implemented by each pane.
<code>PCMDDataHandler</code>	Extended by an editor's custom data handler.

3.1.3.3.1.3 Data Model

The wizard framework, on which the editor framework is based, provides a data model that makes it easier to handle and store the data that administrators enter into the editor.

The data model provides the following benefits:

Creating and Managing Content

- Holds data across requests, so that the data is available to all the panes and to the data handler.
- Data items within the data model can be associated with wizard framework UI controls, so that data entered into the control is automatically saved into the data model, and values stored in the data model are automatically loaded into the controls.

Wizard framework controls are defined in the `com.sapportals.admin.wizardframework.components` package.

The data model is a set of key-value pairs that is available via the editor context, which is exposed as an `IEditorContext` parameter in key methods of the editor framework.

For example, when the data handler's `loadData()` event is called, the editor context is supplied and you can load default values into the data model. The following example saves the value `credit` in the `details.paymentMethod` property of the data model:

```
public void loadData(  
    IEditorContext context,  
    IPrincipal principal,  
    PPLogger logger)  
    throws EditorDataException {  
  
    context.setProperty("details.paymentMethod", "credit");  
}
```

Data Model and UI Controls

To synchronize a wizard framework UI control with a specific value in the data model, call `setValueTargetPath()` on the control and pass the property name in the data model that holds the control's value.

For example, the following creates an input field called *Last Name*, whose value is associated with the editor context property `details.lastName`.

```
TextInputComponent firstName = new TextInputComponent("Last Name");  
firstName.setValueTargetPath("details.lastName");
```

If you change the value in the data model, the control's value automatically changes. The following changes the value of the data model property `details.lastName`, which causes the value to be displayed in the *Last Name* control.

```
context.setProperty("details.lastName", "Smith");
```

HTMLB controls cannot be linked directly to the data model. Store the values of a pane's HTMLB controls into the data model during the `processInput()` method, as described in [Step 1: Creating Panes \[Page 33\]](#).

Loading and Saving Data

You can create a data handler to handle the loading of data into the data model and the saving of the data model into the PCD. A data handler provides the following methods:

- `loadData()`: Loads data into the data model.
- `saveData()`: Stores the data model into the PCD.

For more information on creating a data handler, see [Step 3: Creating a Data Handler \[Page 33\]](#).

Reloading Data

The editor framework calls `loadData()` on the data handler when the editor is first launched, and whenever the user clicks *Refresh*. If you need the data reloaded at any other time, call `setLoadDataRequired()` on the editor context and pass the value `true`.

```
context.setLoadDataRequired(true);
```

Dirty State

To indicate to the editor that data has changed and, therefore, should be saved, set the dirty state to `true`. This enables the *Save* button, and also triggers a dialog for saving changes if the administrator clicks *Close*.

Set the dirty state by calling `setDirty()` on the editor context, as follows

```
context.setDirty(true);
```

3.1.3.3.1.4 Events

The editor can trigger events when administrators perform actions within the editor, such as clicking a button. The following types of events can be triggered:

- [Generic Events \[Page 33\]](#): Events triggered when an administrator clicks a button in the generic toolbar at the top of the editor. These events are triggered and captured automatically by the editor framework.

You cannot change the buttons displayed in the generic events toolbar, but you can create a custom events handler for overriding or extending the standard event handling.

- [Custom Events \[Page 33\]](#): Events, either client or server side, that you trigger and capture.

3.1.3.3.1.4.1 Generic Events

The following lists the generic events that are triggered when the corresponding button is clicked in the generic toolbar at the top of the editor, and describes the default handling of the events:

- **Save:** Executes the `saveData()` method on the editor's data handler. For more information on data handlers, see [Step 3: Creating a Data Handler \[Page 33\]](#).
- **Close:** Closes the editor.

If object properties were changed (that is, the dirty state was set either by the property editor or manually by the editor), the editor framework first displays a dialog asking whether to save the changes. If the administrator clicks **Yes**, the `saveData()` method on the data handler for the editor is called.
- **Refresh:** Calls `loadData()` on the data handler for the editor.
- **Edit Mode:** Calls the `loadData()` on the data handler for the editor and tries to write-enable the editor.

Edit Mode enables you to start editing an object that was opened in read-only mode. The editor may be in read-only mode because the object was locked when the editor was launched, or the administrator did not have permission to edit the object.

Creating and Managing Content

Instead of having to close and reopen the editor, the administrator can click *Edit Mode* to check if the object is now unlocked or permissions were granted to edit the object.

- **Preview** (no default implementation): Calls `handlePreviewEvent()` on the editor's event handler. The default event handler for an editor provides an empty implementation for the preview event handler.

Custom Generic Events Handler

You can override the default handling of the generic events by creating an event handler that either:

- **Extends `GenericEventsHandler`**: You can extend the implementation of an event handler by overriding a handler method, providing your own implementation, and at the end of the method calling the same handler method in the super class.
- **Implements `IGenericEventsHandler`**: You can replace the default implementation of the event handler for the `Refresh`, `Edit Mode` or `Preview` events.

To include a custom event handler, specify the event handler class in the `com.sap.portal.admin.editorframework.eventsHandler` property of the `<component-config>` element for the editor component in the `portalapp.xml`, as shown in the following example:

```
<property name="com.sap.portal.admin.editorframework.eventsHandler"
  value="myEventsHandler"/>
```

3.1.3.3.1.4.2 Custom Events

You can trigger events, either client or server side, using the standard Enterprise Portal Client Framework coding. The following are examples:

- **Client Side**: When creating the UI for a pane, call `setOnClientClick()` on an HTMLB button control and pass the name of a client-side JavaScript function. This function is called when an administrator clicks the button.

```
paymentTypeButton.setOnClientClick("alertSubmit()");
```

- **Server Side**: A server-side event can be triggered in the following ways:
 - When creating the UI for a pane, call `setOnClick()` on an HTMLB button control and pass the name of a server-side event. A server-side event is triggered when an administrator clicks the button.

```
paymentType.setOnClick("coordPayType");
```

- Create a custom toolbar button, as described in [Step 1: Creating Panes \[Page 33\]](#). A server-side event is triggered when an administrator clicks the button.
- Call `raiseServerEvent()` from JavaScript code.

```
function callServerEvent() {
    raiseServerEvent("myServerEvent");
}
```

Capturing Events

Server-side events can be captured by the following:

- **Event Handler Method**: You can create a method to be executed when a server event is triggered. For an event called `event`, `doEvent()` is called.

An event is captured in the same pane in which it was triggered.

Creating and Managing Content

- **Editor Pane:** You can specify that an editor pane be displayed whenever a server-side event is triggered. This provides a way for an editor to navigate between panes without the user selecting a pane radio button.

To specify a pane to be displayed for a server-side event, call `setActivatingEvent()` on the pane object and specify the event that triggers the editor to navigate to that pane. This is done in the editor component's `setUpEditor()` method.

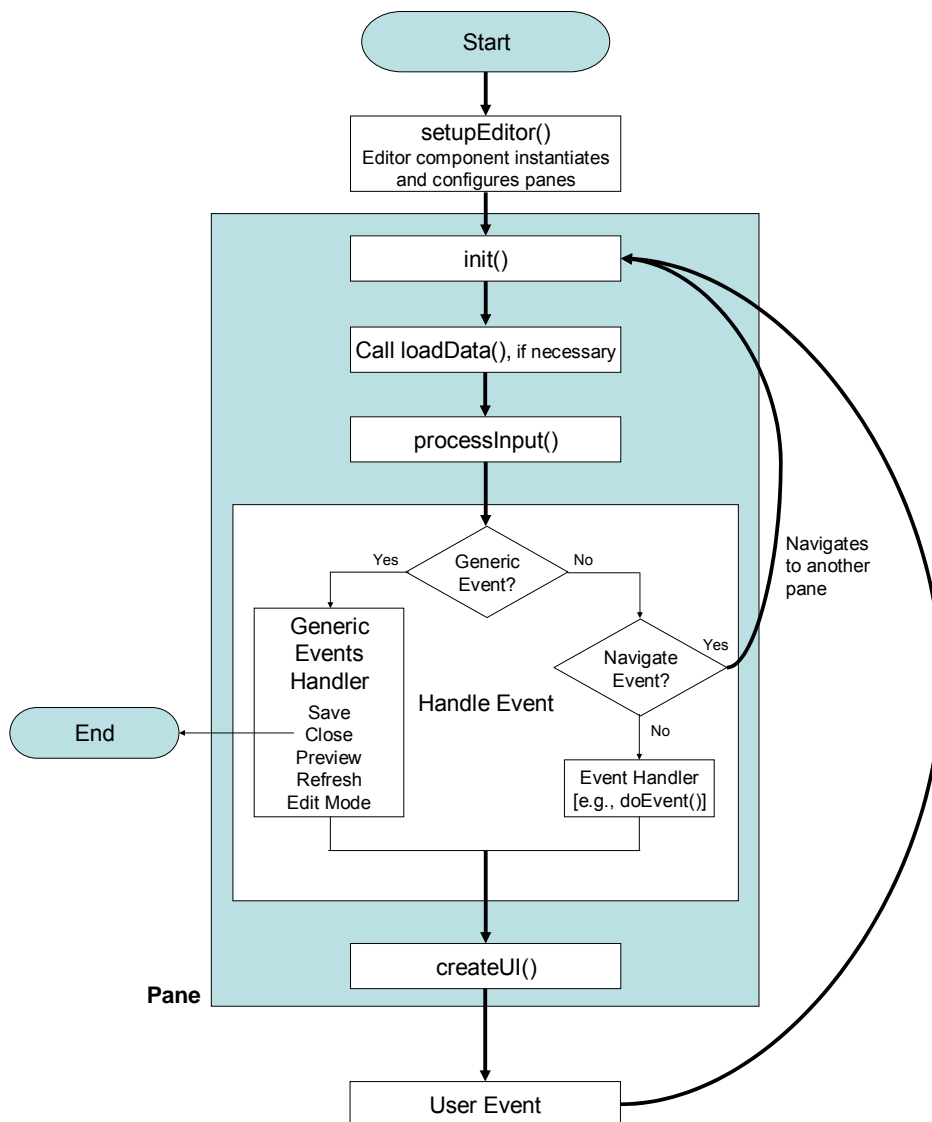
For example, the following specifies that the `personalDetails` server-side event triggers the editor to navigate to the `personalDetails` pane:

```
personalDetailsPane.setActivatingEvent("personalDetails");
```

3.1.3.3.1.5 Process Flow

The following shows the process flow when an editor is launched. The green rectangle includes the steps that occur within the current pane object.

Each user event causes another request, which is handled by the current pane class.



3.1.3.3.1.6 Parameters

This section describes additional parameters in the editor framework.

Editor Parameters

The editor context holds the parameters that were passed to the editor in the URL, including the object ID of the object being edited, editor ID and session ID. To get these parameters, do the following:

- To get a `Map` of all the parameters, call `getInitialParameters()` on the editor context.
- To get the object ID, call `getObjectID()` on the editor context.

Client Parameters

The editor framework enables you to set a client-side parameter from a server-side method, for example, during creation of the user interface for a pane or during the handling of a server-side event.

To create a client-side parameter, call `setClientProperty()` on the editor response and pass the parameter name and value, as shown in the following example, where `context` is the editor context:

```
context.getEditorResponse().setClientProperty("myKey", "myValue");
```

The client-side parameters are available from the `parametersMap` JavaScript object by calling the object's `get()` method, as in the following example:

```
parametersMap.get("myKey")
```

On the client side, you can add additional parameters to `parametersMap` by calling the object's `set()` method – for example, `parametersMap.set("myKey2", "myValue2")` – but these parameters are only available during the current response.

3.1.3.3.2 How to Create an Editor

An editor is a portal application, deployed in a PAR, with the following parts:

- **Editor Component:** A portal component that represents the overall editor.
- **Panes:** Each pane is a Java class that defines the user interface for one section of the editor.
- **Data Handler:** A Java class that defines where to store the data gathered within the editor.

Workflow

The following are the steps required for creating an editor:

1. [Step 1: Creating Panes \[Page 33\]](#)
2. [Step 2: Creating an Editor Component \[Page 33\]](#)
3. [Step 3: Creating a Data Handler \[Page 33\]](#)

3.1.3.3.2.1 Step 1: Creating Panes

An editor contains one or more panes that contain the input fields, buttons and other controls that make up the user interface.

Creating and Managing Content

An editor with a single visible pane displays the title of the pane in the title bar of the editor, just below the toolbar. An editor with more than one visible pane displays in the title bar a radio button group that enables the user to navigate between panes.



When possible, use wizard framework controls for building the pane's user interface. Each of these controls enable you to associate it with an item in the data model so that the wizard framework automatically saves the control's value to the data model.

If the wizard framework does not contain the control that you need, you can use HTMLB controls, but you will have to manually save the control's value to the data model in the `processInput()` method for that pane.

Procedure

For each pane, perform the following:

1. Create a new class that implements `IEditorPane`.
2. Implement `createUI()`, in which you build a user interface with the help of wizard framework and HTMLB controls. The following is the method's signature:

```
public void createUI(IEditorContext context, IWorkspace workspace) {  
}
```

The `IWorkspace` object passed into the method represents the entire editor window.

3. From the workspace, get the `IEditorContainer`, which represents the area of the editor window set aside for your custom user interface.

```
IEditorContainer container = workspace.getEditorContainer();
```

4. Create your user interface with wizard framework and HTMLB controls that you add to the editor's `IEditorContainer` object.

For example, the following adds an input field called `lastNameID`, whose initial value is set to the value stored in the data model property called `details.lastName`, and the field is enabled or disabled based on whether the editor is read only.

```
InputField lastName = new InputField("lastNameID");  
lastName.setValue(ctx.getProperty("details.lastName"));  
container.addComponent("Last Name", lastName);  
lastName.setEnabled(!ctx.isReadOnly());
```

For more information on HTMLB controls, see [HTML-Business for Java \[Page 33\]](#).

5. Implement the following `IEditorPane` methods, which are called on every request for the pane:

Creating and Managing Content

- **init()**: Perform any initialization for the pane.
- **getEditorResources()**: Load any resources for the pane, such as JavaScript files or style sheets.
- **processInput()**: Store user input from HTMLB controls in the editor context. The editor context and the HTMLB page context are passed into the method.

For example, for an HTMLB input control called *lastNameID*, the following stores the user input in the editor context property called *details.lastName*:

```
context.setProperty("details.lastName",
    pageContext.getDataForComponentId("lastNameID").
        getValueAsString());
```

User input for wizard framework controls are automatically stored in the editor context.

Options

The following lists additional steps that you can perform within the pane class:

- **Set Read-Only State:** You can determine if the editor and, therefore, the pane should be read only, and then set the editor context to read only for the current request, as follows:

```
context.setReadOnly();
```

- **Navigate to Another Pane:** To navigate to another pane, trigger the server event that was set for that pane. For example, the following creates an HTMLB button called *creditCardID* whose *onClick* event raises the *creditPane* event:

```
Button creditCard = new Button("creditCardID");
creditCard.setOnClick("creditPane");
```

This causes the *creditPane* server event to be triggered. The editor navigates to the credit pane if the credit pane's activating event was set to *creditPane*, as follows:

```
creditCard.setActivatingEvent("creditPane");
```

For more information on creating instances of panes and setting the activating event, see [Step 2: Creating an Editor Component \[Page 33\]](#).

For more information on events, see [Events \[Page 33\]](#).

- **Create a Custom Toolbar:** You can create a toolbar of custom buttons at the bottom of the pane. The following example shows how to create a custom toolbar button:

```
Action action = new Action("myAction", "My Action");
workspace.getToolBar().add(action);
```

When an administrator clicks the custom button, a server-side event is triggered whose name is the first parameter in the constructor for the button. In the above example, an event called *myAction* is triggered, and the event is handled by the pane's *doMyAction()* method.

3.1.3.3.2 Step 2: Creating an Editor Component

Each editor contains an editor component, whose Java class extends *AbstractEditorComponent*, a descendent of *AbstractPortalComponent*, making the

Creating and Managing Content

editor a portal component that can be displayed as an View in the administration pages of the portal.

`AbstractEditorComponent` implements `IEditorBuilder`, which defines only one method, `setupEditor()`, which is passed an `IEditorConfiguration` object. In this method, the editor does the following:

- Creates instances of the panes as `PaneMetaData` objects.
- Adds the panes to the `IEditorConfiguration` object.
- Sets the default pane (that is, the pane that is displayed when the editor is launched).

Procedure

1. Create a new class that extends `AbstractEditorComponent`.
2. Implement `setupEditor()`, in which you will create instances of the panes, configure them, and add them to the editor. The following is the method's signature:

```
public void setupEditor(
    IEditorConfiguration editorConfig,
    IPortalComponentProfile profile,
    Map params,
    ResourceBundle editorTexts) {
}
```

3. Create instances of the panes for the editor and configure the panes. For each pane, do the following:
 - a. Create a `PaneMetaData` object for representing the pane. The `PaneMetaData` constructor takes the `Class` object of your pane class, and a string that is the name of the pane.

```
PaneMetaData personalDetailsPane =
    new PaneMetaData(PersonalDetailsPane.class, "Personal Info");
```

- b. Set the tooltip displayed when hovering over the radio button that displays the pane.

```
personalDetailsPane.setDescription("personal details information");
```

- c. Set the server event that causes the pane to be displayed.

```
personalDetailsPane.setActivatingEvent("personalDetails");
```

A pane can also be displayed by the user by clicking the pane's radio button at the top of the editor.

- d. Indicate whether a radio button is displayed for this pane in the pane selection area at the top of the editor.

```
personalDetailsPane.setPaneSelectionMember(true);
```

If no radio button is displayed for the pane, the only way to display the pane is via an activating event.

4. Add each pane to the editor by calling `addEditorPane()` on the `IEditorConfiguration` object that is passed into the `setupEditor()` method.

```
editorConfig.addEditorPane(personalDetailsPane);
```

5. Specify the default pane, that is, the pane that is displayed when the editor is launched.

```
editorConfig.setDefaultEditorPane(paymentPane);
```

Creating and Managing Content

6. Create an entry in the `portalapp.xml` for the editor component. The `<component>` element is similar to a standard component, except that you specify the following in the `<component-profile>` element:
 - **Data Handler:** In the `com.sap.portal.admin.editorframework.dataHandler` property, specify the class or service that implements the data handler.

For more information on creating the data handler, see [Step 3: Creating a Data Handler \[Page 33\]](#) and [portalapp.xml \[Page 33\]](#).
 - **JavaScript Resource:** If you are including a JavaScript resource for the editor, specify in the `com.sap.portal.admin.editorframework.jsResource` property the path to the JavaScript file.
 - **Generic Events Handler:** If you are including a custom generic events handler, specify in the `com.sap.portal.admin.editorframework.eventsHandler` property the class that implements the events handler.

For more information on creating custom generic events handlers, see [Generic Events \[Page 33\]](#).
 - **iView Properties:** Specify iView properties in order to properly display the editor. For example, remove the iView tray and set the isolation mode to URL.

The following is an example `<component>` element for an editor component:

Creating and Managing Content

```

<components>
  <component name="TrainEditor">
    <component-config>
      <property name="ClassName" value="myPackage.TrainEditor"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name=
        "com.sap.portal.admin.editorframework.dataHandler"
        value="myEditor.myDataHandler"/>
      <property name="com.sap.portal.iview.ShowTray"
value="false"/>
      <property name="com.sap.portal.reserved.iview.IsolationMode"
        value="URL"/>
      <property name="com.sap.portal.iview.HeightType"
        value="FULL_PAGE"/>
      <property name="isStateless" value="false" />
    </component-profile>
  </component>
</components>

```

Result

The following is an example of `setupEditor()` for creating and configuring two panes:

```

public void setupEditor(
    IEditorConfiguration editorConfig,
    IPortalComponentProfile profile,
    Map params,
    ResourceBundle editorTexts) {

    //Create and configure personal info pane.
    PaneMetaData personalDetailsPane =
        new PaneMetaData(PersonalDetailsPane.class, "Personal Info");
    personalDetailsPane.setDescription("personal details
information");
    personalDetailsPane.setActivatingEvent("personalDetails");
    personalDetailsPane.setPaneSelectionMember(true);

    //Create and configure the payment info pane.
    PaneMetaData paymentPane =
        new PaneMetaData(PaymentPane.class, "Payment Info");
    paymentPane.setPaneSelectionMember(true);
    paymentPane.setActivatingEvent("paymentPane");

    //Add panes to the editor.
    editorConfig.addEditorPane(personalDetailsPane);
    editorConfig.addEditorPane(paymentPane);

    //Set default pane.
    editorConfig.setDefaultEditorPane(PersonalDetailsPane.class);
}

```

3.1.3.3.2.3 Step 3: Creating a Data Handler

You can create a custom data handler for your editor for loading initial values into the editor controls and saving the values entered by the user to the PCD. A data handler, which extends `PCMDDataHandler`, implements `loadData()` and `saveData()`.

Creating and Managing Content

The `loadData()` method is called when the editor is launched, when the editor is refreshed (that is, when the administrator clicks *Refresh*) and when the editor manually requests a reload of data by calling `setLoadDataRequired()` on the editor context.

The `saveData()` is called when the editor framework tries to save the editor's data, for example, when the user clicks *Save* in the editor toolbar.

If you do not create a data handler for your editor, the default data handler is called. The default handler does not load any data and only saves changes made to the property editor.

Procedure

1. Create a new class that extends `PCMDDataHandler`.
2. Implement `loadData()`, in which you initialize the data model by setting properties in the editor context. The context is passed into the method as an `IEditorContext` object.

The following is the method's signature:

```
public void loadData(
    IEditorContext context,
    IPrincipal principal,
    PPLogger logger)
    throws EditorDataException {
}
```

In the editor context, set the value for properties that later will be saved to the PCD for the currently edited PCD object, as in the following example:

```
context.setProperty("payment.type", "creditCard");
```

3. Implement `saveData()`, in which you save properties from the editor context into the PCD. The following is the method's signature:

```
public void saveData(
    IEditorContext context,
    IPrincipal principal,
    PPLogger logger)
    throws EditorDataException, EditorResourceException {
}
```

In `saveData()`, do the following:

- **Save Data from the Editor:** The data entered by the user in your editor is stored in the data model in the editor context. Save this data to the PCD, that is, save each item in the data model to a specific attribute of the PCD object that is being edited.

The following is an example of looking up the current PCD object and saving an object attribute:

```
//Get the object ID for the currently edited PCD object.
Map parametersMap = context.getInitialParameters();
String objId = (String)parametersMap.get("objectID");

//Set environment variables for the PCD lookup.
Hashtable env = new Hashtable();
env.put(Context.SECURITY_PRINCIPAL, principal);
env.put(Constants.APPLY_ASPECT_TO_CONTEXTS,
    Constants.APPLY_ASPECT_TO_CONTEXTS);
env.put(Constants.REQUESTED_ASPECT,
```

Creating and Managing Content

```

        PcmConstants.ASPECT_ADMINISTRATION);

//Perform the lookup, returning an IAdminBase object.
InitialContext iCtx;
IAdminBase adminBase = null;
try {
    iCtx = new InitialContext(env);
    adminBase = (IAdminBase) iCtx.lookup(objId);
} catch (NamingException ne) {
}

//Get the object's IAttributeSet interface.
IAttributeSet attrSet = (IAttributeSet)
    adminBase.getImplementation(IAdminBase.ATTRIBUTE_SET);

//Set the attribute.
attrSet.putAttribute("myProperty", "200");

//Save the changes.
try {
    attrSet.save();
} catch (ValidationException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

```

- **Save Changes Made in the Property Editor:** Call `saveData()` on the data handler's super class, as follows:

```
super.saveData(context, principal, logger);
```

This saves any changes made to the object via the property editor.

4. If you want the data handler to be available to editors in other portal applications (PARs), implement the `IService` interface.

Retrieving Values from the Property Editor

From the data handler's `getModifiedPropertyObject()` method, you can retrieve the value of any property that is displayed in the property editor, as in the following example:

```

IProperty prop = this.getModifiedPropertyObject(context).getProperty(
    "com.sap.portal.pcm.Description");
String description = prop.getValue();

```

3.1.3.3.3 Essential Information

This section provides the following information to help you manage systems:

- [portalapp.xml \[Page 33\]](#): Information about the `portalapp.xml` file for an editor application.
- [JARs and Packages \[Page 33\]](#): The packages and JAR files required for creating an editor.

3.1.3.3.1 portalapp.xml

The following is a sample portalapp.xml for an editor:

```
<application>
  <application-config>
    <property name="releasable" value="false"/>
    <property name="Vendor" value="sap.com"/>
    <property name="SecurityArea" value="NetWeaver.Portal"/>
    <property name="SharingReference" value=
      "com.sap.portal.admin.wizardframework,
      com.sap.portal.admin.editorframework, com.sap.portal.htmlb" />
  </application-config>

  <components>
    <component name="TrainEditor">
      <component-config>
        <property name="ClassName" value="TrainEditor"/>
        <property name="SafetyLevel" value="low_safety"/>
      </component-config>
      <component-profile>
        <property name=
          "com.sap.portal.admin.editorframework.dataHandler"
          value="myEditor.myDataHandler"/>
        <property name="com.sap.portal.iview.ShowTray"
          value="false"/>
        <property name="com.sap.portal.reserved.iview.IsolationMode"
          value="URL"/>
        <property name="com.sap.portal.iview.HeightType"
          value="FULL_PAGE"/>
        <property name="isStateless" value="false" />
      </component-profile>
    </component>
  </components>

  <services>
    <service name="myDataHandler">
      <service-config>
        <property name="className" value="TrainDataHandler" />
        <property name="SafetyLevel" value="medium_safety" />
        <property name="startup" value="false" />
      </service-config>
    </service>
  </services>
</application>
```

Application Configuration Element

In the <application-config> element, add a SharingReference for the following services:

Service/Component	Sharing Reference
Wizard Framework	com.sap.portal.admin.wizardframework
Editor Framework	com.sap.portal.admin.editorframework
HTMLB	com.sap.portal.htmlb

```
<application-config>
  <property name="releasable" value="false"/>
  <property name="Vendor" value="sap.com"/>
```

Creating and Managing Content

```

    <property name="SecurityArea" value="NetWeaver.Portal"/>
    <property name="SharingReference" value=
        "com.sap.portal.admin.wizardframework,
        com.sap.portal.admin.editorframework, com.sap.portal.htmlb"
    />
</application-config>

```

Components Element

An editor contains one portal component, the editor component, which is based on a class that extends `AbstractEditorComponent`. For more information on creating the editor component's `<component>` element, see [Step 2: Creating an Editor Component \[Page 33\]](#).

```

<components>
  <component name="TrainEditor">
    <component-config>
      <property name="ClassName" value="myPackage.TrainEditor"/>
      <property name="SafetyLevel" value="low_safety"/>
    </component-config>
    <component-profile>
      <property name=
        "com.sap.portal.admin.editorframework.dataHandler"
        value="myEditor.myDataHandler"/>
      <property name="com.sap.portal.iview.ShowTray"
value="false"/>
      <property name="com.sap.portal.reserved.iview.IsolationMode"
        value="URL"/>
      <property name="com.sap.portal.iview.HeightType"
        value="FULL_PAGE"/>
      <property name="isStateless" value="false" />
    </component-profile>
  </component>
</components>

```

Services Element

If you want the data handler to be available to editors in other portal applications (PARs), you must expose the data handler as a portal service and list it in the `<services>` element. The data handler class must implement `IService`.

If the data handler is exposed as a service, the name of the data handler service – instead of the class – is specified in the `com.sap.portal.admin.editorframework.dataHandler` property in the `<config-profile>` section for any editor component that wants to use it, as described in *Component Element* above.

```

<services>
  <service name="myDataHandler">
    <service-config>
      <property name="className" value="TrainDataHandler" />
      <property name="SafetyLevel" value="medium_safety" />
      <property name="startup" value="false" />
    </service-config>
  </service>
</services>

```

3.1.3.3.2 JARs and Packages

Packages

- `com.sapportals.admin.wizardframework.components`
- `com.sap.portal.admin.editor`
- `com.sapportals.htmlb`
- `com.sapportals.common` (for logger)

JARs

- `com.sap.portal.admin.wizardframework_api.jar`
- `com.sap.portal.admin.editorframework_api.jar`

3.1.4 Client-Side Eventing

Purpose

The *Enterprise Portal Client Framework* (EPCF) provides an infrastructure for scripting used in iViews and by the portal.

To keep the application design simple and maintain compatibility to different browsers, Web applications usually avoid scripting. However, there are tasks that make it necessary to use scripting, such as:

- Increasing user acceptance for example with context sensitive entry helpers
- Enhancing response time of the application for example through validation of input values

When a business application uses more than one iView, you need the EPCF service to transfer data between the iViews. The EPCF service provides:

- Mechanisms for eventing between iViews.
- A Java object, called a client data bag, that serves as transient data buffer on the browser.

The EPCF implementation itself is based on JavaScript and Java applets.

3.1.4.1 EPCF Levels

The EPCF level defines which functionality of the EPCF service is available to the Web application. The EPCF level affects the size of the generated HTML data is generated, that is sent to the client. A higher level generates more HTML data. The EPCF has the following levels.

0: No EPCF Service

This level generates no JavaScript or Java applet framework functions. Communication between iViews is not possible.

1: JavaScript

This level generates framework functions for JavaScript.

2: JavaScript and Applet

This level generates framework functions for JavaScript and Java applet.

Detailed feature list

Feature	Level	Description
Implicit softening the JavaScript Origin Policy [Page 33]	1	Enables scripting between <i>IFrames</i> from the same domain but different hosts.
System function	1	Retrieves the EPCF version, the current EPCF level and client information.
Client eventing	1	Client communication service for iViews.
Predefined Client Events	1	Collection of useful client events.
<i>Client Data Bag</i>	1* / 2	Buffer for the client that stores JavaScript data as long as the portal session exists.

3.1.4.2 EPCF API

The EPCF service defines the *Enterprise Portal Client Manager* (EPCM) JavaScript object. With the methods of the EPCM object you can access the EPCF service functions as follows:

```
EPCM.[API_method_name] ( [Params]* );
```

iViews can access the EPCM object from every portal page or *IFrame*. Every iView of the *Portal Runtime* (PRT) contains the EPCM object. As a result every embedded or isolated iView can use the EPCF service with the method:

```
EPCM.subscribeEvent(, eventName, eventHandler );
```

For details on namespaces, see section [Namespaces \[Page 33\]](#) in the glossary.

Every EPCM object stores the data it receives and delegates them automatically to the registered EPCM objects.

With EPCF level 2 every portal page and as a result in all isolated iViews, contain a Java applet. The applet serves as a class factory for methods that return references to the intrinsic classes. These classes are implemented as *Singletons* therefore every class has only a single instance in a portal session, even when the iViews are reloaded. The applet object is instantiated inside every portal page frame.

The EPCF API has following parts:

- System API
The [system API \[Page 33\]](#) provides methods to get the version and the level of the EPCF service and information about the client.
- Event API
The [event API \[Page 33\]](#), which allows iViews to communicate with each other and with the portal environment itself on the. This is done by using JavaScript functions on the client which are invoked on client events like *onload*, *onclick* and so on.
- Client Data Bag API

Creating and Managing Content

The [client data bag API \[Page 33\]](#) offers a transient data buffer for iViews. The data remains in the buffer are saved even after the iView or the whole portal page is reloaded. Depending on the EPCF level, following storage mechanisms are used:

- EPCF level 1 - JavaScript
Values are stored as cookies in the browser.
- EPCF level 2 – Java Applet
Values are stored as a Java class attribute.
- WorkProtect API
The [WorkProtect API \[Page 33\]](#) provides the infrastructure for handling unsaved data in a *stateful* application.
- Navigation API
The [Navigation API \[Page 33\]](#) provides methods to navigate in the portal.
- EPCM Proxy
The [EPCM proxy API \[Page 33\]](#) enables the EPCF functions in portal applications that are rendered in their own *IFrame* (for example, ITS-based applications and BSP).

3.1.4.2.1 System API

With the EPCF system API you have access to the settings of the EPCF service.

EPCM.getVersion()

This method returns the current framework version as type *number*.

Usage

```
<script language="JavaScript">  
    var version = EPCM.getVersion();  
</script>
```

EPCM.getLevel()

This method returns the current EPCF level as type *number*.

The EPCF level defines which EPCF services are available. The portal application has to take care, that it uses services which are available at the current EPCF level.

Usage

```
<script language="JavaScript">  
    if (EPCM.getLevel() >= 2) {  
        EPCM.storeClientData( "urn:com.sap.myObjects",  
                             "person", "Albert Borland" );  
    }  
</script>
```

EPCM.getUAType()

This method returns the client type as type *number*.

Usually this method will be used together with the method `getUAVersion()` and `getUAPlatform()`. The return value can be compared to predefined EPCM-constants:

**EPCM.MSIE, EPCM.NETSCAPE, EPCM.MOZILLA, EPCM.OPERA, EPCM.NOKIA,
EPCM_UP, EPCM_ERICSSON, EPCM_MSPIE, EPCM_PALM EPCM.OTHER.**

Usage

```
<SCRIPT language ="JavaScript">
    if(EPCM.getUAType() == EPCM.MSIE){/*codingfor      IE*/ }
    if(EPCM.getUAType() == EPCM.MOZILLA){/*codingfor
Mozilla*/ }
</SCRIPT>
```

EPCM. getUAVersion()

This method returns version of the client as type *number*.

Usage

```
<SCRIPT language ="JavaScript">
    if(EPCM.getUAType() == EPCM.MSIE){
        if(EPCM.getUAVersion() ==      5.0){/*codingfor MSIE 5.0
*/ }
        if(EPCM.getUAVersion() > 5.5){/*codingfor MSIE 5.0+*/ }
    }
</SCRIPT>
```

EPCM.getUAPlatform()

This method returns the platform on which the client is running as type *number*.

The return value can be compared to predefined EPCM constants:

**EPCM.NT_PLATFORM, EPCM.WIN_PLATFORM, EPCM.MAC_PLATFORM,
EPCM.LINUX_PLATFORM, EPCM.WAP_PLATFORM, EPCM.PDA_PLATFORM,
EPCM.OTHER_PLATFORM.**

Usage

```
<SCRIPT language ="JavaScript">
    if(EPCM.getUAPlatform() == EPCM.LINUX_PLATFORM){
        /* coding that will only be processed if the client runs
```

```
LINUX */  
}  
</SCRIPT>
```

getInstanceId() Method

This method returns an unique EPCF instance as type *String*.

The method is used by the EPCF core to distinguish the pages after a page refresh.

Usage

```
<SCRIPT language ="JavaScript">  
    document.write("EPCMInstanceId = " + EPCM.getInstanceId() );  
</SCRIPT>
```

EPCM.getUniqueWindowId()

This method returns an unique identifier of the *IFrame* as type *String*.

You can use this method to append the returned *IFrame* identifier string to the name you use to define a *client data bag*. This creates a *client data bag* that can only be accessed by a specific *IFrame*.



The method returns *null* when the object returned by *window.top* is not accessible because of security reasons ([JavaScript origin policy \[Page 33\]](#)).

Usage

```
<SCRIPT language ="JavaScript">  
    document.write("WindowId = " + EPCM.getUniqueWindowId() );  
    ...  
    ECPM.storeClientData(  
  
        "com.sap.portal:test",EPCM.getUniqueWindowId()+"Item",myItem);  
    ...  
</SCRIPT>
```

3.1.4.2.2 Event API

The EPCF event API provides methods for the event handling on the client.

EPCM.subscribeEvent(namespace, eventName, eventHandler)

This method assigns an event handler to the specified event.

Creating and Managing Content

The method sets the event handler to the subscription list for the event defined by the *nameSpace* and the *eventName*. The combination of *nameSpace*, *eventName* and *eventHandler* must be unique. It is not possible to register the same event handler to several events. See section [Namespaces \[Page 33\]](#) for more details.



Isolated iViews have to subscribe on every page *Refresh* or *Reload*.

Parameter Description

Parameter	Type	Description
nameSpace	String	URN [Page 33] of the event namespace.
eventName	String	The event name you subscribe to. You can use an asterisk (*) to subscribe for all events of this namespace.
eventHandler	Function	Reference to the event handler.

Usage

```
<script language ="JavaScript">
    function onWakeup( eventObj ) {
        alert( "got a wakeup call from " +
            eventObj.sourceId + ": " + eventObj.dataObject );
    }
    ...
    EPCM.subscribeEvent( "urn:com.sap:alarmClock",
        "morningCall", onWakeup );
    ...
    EPCM.subscribeEvent( "urn:com.sap:alarmClock", "*",
onWakeup );
</script>
```

EPCM.raiseEvent(nameSpace, eventName, dataObject [, sourceId])

This method raises the event defined by *nameSpace* and *eventName*. The EPCF service calls all event handlers which are registered for this event and passes the *event object* on to the event handler.

The *event object* is created by the EPCF service whenever an event is raised. It combines the *dataObject*, the *eventName* and the *sourceId* (which may be null) to a single argument for the event handler.

Parameter Description

Parameter	Type	Description
-----------	------	-------------

Creating and Managing Content

nameSpace	String	URN [Page 33] of the event name-space.
eventName	String	The event name with which the event is raised..
dataObject	Object	An object (<i>String</i> , <i>Number</i> , <i>Boolean</i> or <i>Object</i>) that contains a description of the event.
sourceId (optional)	String	The component id of the event source, for example, the id defined at design-time. Specify <i>Null</i> or no parameter if you do not need the id.

Usage

```

<SCRIPT language ="JavaScript">
    ...
    EPCM.raiseEvent( "urn:com.sap:alarmClock", "morningCall",
        "Good morning ladies and gentlemen", "iView_0815" );
    ...
</SCRIPT>

```

3.1.4.2.3 Client Data Bag API

The EPCF *client data bag* API provides methods to store data in a transient data buffer on the client.

EPCM.storeClientData(nameSpace, name, value)

This method saves data in *value* under a key. The key is generated by combining the parameters *nameSpace* and *name*. If the key already exists, the stored data will be overwritten.

Parameter Description

Parameter	Type	Description
nameSpace	String	URN [Page 33] for the first part of the key under which the data is stored. <i>nameSpace</i> will be combined with <i>name</i> .
name	String	This name, combined with <i>namespace</i> , creates the key under which the data is stored.
value	String	Data to be stored.



The parameter value must be of type *String*. Primitive data types will be converted to *String*, complex data types and references are not supported. This restriction is necessary to guarantee that the *client data bag* functions are working in a JavaScript environment using the browser cookies for clients that have no Java support.

Usage

```
<SCRIPT language ="JavaScript">
    var selectedPerson = "Tim Taylor"
    EPCM.storeClientData( "urn:com.sap.myObjects", "person",
        selectedPerson );
</SCRIPT>
```

EPCM.loadClientData(nameSpace, name)

This method returns the data stored under the specified key as *String*. The key is generated by a combination of the parameters *nameSpace* and *name*. If the key does not exist, the method returns *null*.

Parameter Description

Parameter	Type	Description
nameSpace	String	URN [Page 33] for the first part of the key from which the data is reloaded. <i>nameSpace</i> will be combined with <i>name</i> .
name	String	This name, combined with <i>namespace</i> , is the key from which the data is reloaded.

Usage

```
<SCRIPT language ="JavaScript">
    var person=EPCM.loadClientData("urn:com.sap.myObjects",
    "person");
    if ( person != null ){
        /* process person */
    }
</SCRIPT>
```

deleteClientData(nameSpace, name)

This method deletes the data stored under the specified key and the key itself. The key is generated by a combination of the parameters *nameSpace* and *name*.

Parameter Description

Parameter	Type	Description
nameSpace	String	URN [Page 33] for the first part of the key which is. <i>nameSpace</i> will be combined with <i>name</i> .
name	String	This name, combined with <i>namespace</i> , is the key from which is deleted.

Usage

```
<SCRIPT language ="JavaScript">
    EPCM.deleteClientData( "urn:com.sap.myObjects", "person" );
</SCRIPT>
```

3.1.4.2.4 WorkProtect API

The EPCF *WorkProtect* API provides methods to get the status about unsaved data on the page.

EPCM.setDirty(indicator)

This method sets the status of the *dirty indicator* to *true* or *false*.

Parameter Description

Parameter	Type	Description
indicator	boolean	Status of the dirty indicator: <i>true</i> : Page contains unsaved data. <i>false</i> : Page is clean – no unsaved data.

Usage

```
<SCRIPT language ="JavaScript">
    if (storedValue != enteredValue){
        changedData["DataKey"] = enteredValue;
        EPCM.setDirty( true );
    }

    // do other actions ...

    storeArrayToDataBase(changedData);
    EPCM.setDirty( true );

</SCRIPT>
```

EPCM.getDirty()

This method returns the current setting of the *dirty indicator* as type *boolean*. The *WorkProtect* feature uses this method to get the *dirty indicator* for the entire portal page.

Usage

```
<SCRIPT language ="JavaScript">
    var isDirty = EPCM.getDirty( );
```

Creating and Managing Content

```

        alert("Component " + (isDirty) ? "clean" : "dirty" );
    </SCRIPT>

```

EPCM.getGlobalDirty()

This method returns the current setting of the *dirty indicator* as type *boolean*. The difference to the `getDirty()` method is, that the `getGlobalDirty()` method checks the *dirty flag* of all *iViews* on the page and returns a *true* value if at least one of the *iViews* had a *dirty flag* set to *true*, and *false* otherwise.

Usage

```

<SCRIPT language ="JavaScript">
    var isDirty = EPCM.getGlobalDirty( );
    alert("One component " + (isDirty) ? "clean" : "dirty" );
</SCRIPT>

```

3.1.4.2.5 Navigation API

The Navigation API provides the methods to navigate inside the portal. Refer to section [Enterprise Portal Navigation \[External\]](#) for more details about the *navigation service*.

Absolute Navigation

For an *absolute navigation* you have to know the full path name of the component. The full path name starts at the navigation hierarchy root node all the way to the *navigation target*.

EPCM.doNavigate(String target, [int mode, String winFeat, String winName, int history, String targetTitle, String context])

This method triggers the *absolute navigation* on the client. By default, when the parameter *mode* is not specified, the *dirty indicator* of the component is checked by the *WorkProtect* feature and the target is opened in a new window or on the current portal page depending on the result of the check.

The optional parameters are new in Enterprise Portal 6.0 and can be used when the target is displayed in the new window.

The method always returns *false*, for easier use with event handlers like *onClick*.

Parameter Description

Parameter	Type	Description
target	String	Navigation target that corresponds to the location in PCD or another structure (see details below)
mode (optional)	int	0 or not specified: Depending on the setting of the <i>WorkProtect</i> feature the target is opened in a new window or on the current desktop.

Creating and Managing Content

		<p>1: Open target in a new window, with no a portal header and navigation bar.</p> <p>2: Open target in a new window, with a portal header and navigation bar.</p>
winFeat (optional)	String	<p>Window feature string when the target is to be opened in the new window. This is a comma separated list of features with no blanks that has the same syntax as the JavaScript method window.open.</p> <p>Example: "width=400,height=500".</p>
winName (optional)	String	Window title for when the target is opened in a new window.
history (optional)	int	<p>history mode</p> <p>0: Track history entries and allow duplicates.</p> <p>1: Track history entries and do not allow duplicates.</p> <p>2: Do not track history entries.</p>
targetTitle	String	Title for the page title bar. In case the <i>navigation target</i> is sent through an <i>integrator</i> , the title will be the <i>integrator</i> title. You can specify a specific title for this navigation and optional for the history entry.
Context	String	Navigation context URL.

Usage

```

<SCRIPT language ="JavaScript">
    // navigate.
    EPCM.doNavigate(

    "ROLES://portal_content/folder1/role1/workset1/iView111")
</SCRIPT>

<A HREF="myLink"
    onclick="return EPCM.doNavigate

    ('ROLES://portal_content/folder1/role1/workset1/iView111') ">
This is an HTML Link
</A>

```

Result

This starts the navigation to the iView 111 under *role* 1.

Creating and Managing Content

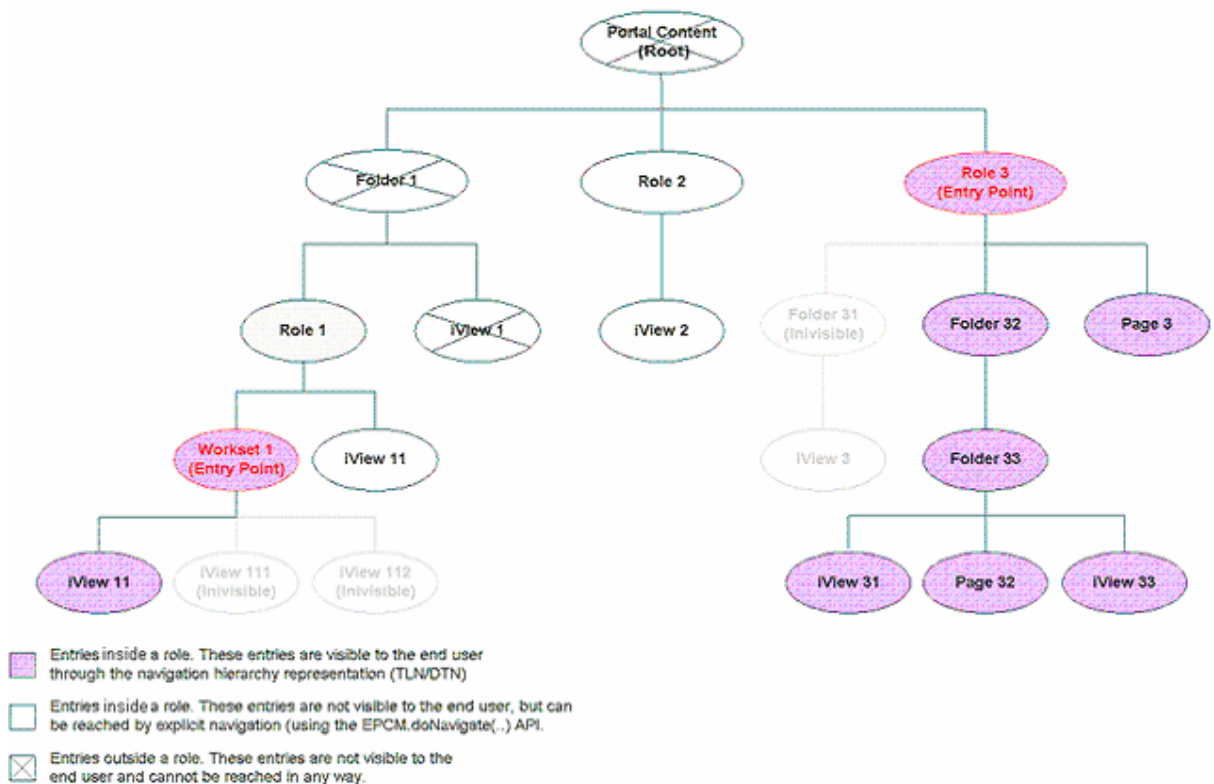


Figure 1: Navigation hierarchy example

Relative Navigation

For a *relative navigation* you specify the relative location of the *navigation target* to the current *navigation node*.

EPCM.doRelativeNavigate(String basenodename, int level, List pnamesList, int mode, String winFeat, String winName, int history, String targetType, String context)

This method triggers the *relative navigation* on the client. You have to know the location of the *navigation target* relative to the current node. That the *navigation model* can create an absolute path, you have to provide at least the `level` or the `pnameslist` parameters in addition to the `basenodename` parameter.

The optional parameters are new in EP 6.0 and can be used when the target is displayed in the new window.

The method always returns *false*, for easier use with event handlers like *onClick*.

Parameter Description

Parameter	Type	Description
<code>basenodeName</code>	String	Current presented URL – current node.

Creating and Managing Content

level	int	Number of hierarchy levels to <i>step up</i> .
pnameslist	List	A list with all the atomic names of the children nodes, relative to the node that has been reached by <i>stepping up</i> the number of levels defined by parameter <code>level</code> .
mode	int	0 or not specified: Depending on the setting of the <i>WorkProtect</i> feature the target is opened in a new window or on the current desktop. 1: Open target in a new window, with no a portal header and navigation bar. 2: Open target in a new window, with a portal header and navigation bar.
winFeat (optional)	String	Window feature string when the target is to be opened in the new window. This is a comma separated list of features with no blanks that has the same syntax as the JavaScript method window.open . Example: "width=400,height=500".
winName (optional)	String	Window title for when the target is opened in a new window.
history (optional)	int	history mode 0: Track history entries and allow duplicates. 1: Track history entries and do not allow duplicates. 2: Do not track history entries.
targetTitle	String	Title for the page title bar. In case the <i>navigation target</i> is sent through an <i>integrator</i> , the title will be the <i>integrator</i> title. You can specify a specific title for this navigation and optional for the history entry.
Context	String	Navigation context URL.

Usage

```

<SCRIPT language ="JavaScript">
    // navigate.
    EPCM.doRelativeNavigate(
        "ROLES:// portal_content/role3/Folder32/Folder33", 2,
        {"page3"}, ..., ..., ...);
</SCRIPT>

<A HREF="myLink"
    onclick="return EPCM.doRelativeNavigate
        ('ROLES:// portal_content/role3/Folder32/Folder33', 2,
        {"page3"}, ..., ..., ...)">

```

```
This is an HTML Link
</A>
```

Result

This starts the navigation from *folder 33* under *role 3* to *page 3* under *role 3*. See figure 1.



If you do not provide the first parameter, the current *navigation node*, the *navigation model* will find the current *navigation node* itself and add it to the path automatically.

Object Based Navigation

The *object based navigation* allows navigation based on actual business objects from productive back end systems. The *object based navigation* is based on the concept of business objects, that perform certain operations and iViews that can be declared as *implementers* of these operations. Every operation has a priority. When choosing the link of the *object based navigation*, the operation with the highest priority, that has an implementing iView in the user role (the default), will be executed.

EPCM.doObjBasedNavigate(String systemAlias, String businessObjName, String objValue, String operation)

This method allows navigating in a context environment, without a specific URL for a *navigation target*. For more details, see the *Object Base Navigation* description.

The method always returns *false*, for easier use with event handlers like *onClick*.

Parameter Description

Parameter	Type	Description
<code>systemAlias</code>	String	The system alias of the business object.
<code>businessObjName</code>	String	Business object name for which the operation was defined.
<code>objValue</code>	String	Any data that has to be transferred to the <i>navigation target</i> when the visualization iView represents relative data. The <code>objValue</code> can be any string that is added to the URL of the <i>navigation target</i> (after the "?" separator) and the target iView can access the <code>objValue</code> via the iView <code>request</code> object.
<code>operation</code>	String	Operation that should be performed when the business object has more than one operation.



The parameters `businessObjName` and `systemAlias` define the *MetaMatrix* name of the *object based navigation* business object.

Creating and Managing Content

If the *object based navigation* uses *relation resolving*, the `objValue` parameter is used to transfer the `HRNPLink`.

3.1.4.2.6 EPCM Proxy

The EPCM proxy enables the EPCF functions for portal applications that are rendered in their own *IFrame* (for example, ITS-based applications and BSP).

Reload Function and Event-Subscription

The EPCF event methods are used by *function reference*. Since the references are kept across the *IFrame* borders, the references become invalid whenever the *IFrame* content is reloaded. To solve this problem, you have to use the second signature of the `EPCM.subscribeEvent` method that references to the current window object.

```
EPCM.subscribeEvent(nameSpace, name, window_reference,
method_name)
```

External applications (for example, BSP, BW-Reports) are rendered in their own *IFrame*. The EPCM Object therefore can convert the event handler registration from `[window_reference,method_name]` to `[iframe_name,method_name]`. With this conversion the method keeps the name and not the object/method reference. When the *IFrame* content is reloaded now, the `iframe_name` and `method_name` are still valid and the event handler, that is located inside the *IFrame*, can be called from the EPCM event manager outside the *IFrame* using the following call:

```
window.frames[iframe_name][method_name]( event_data )
```

IncludeProxy

To simplify the implementation, the `EPCMPROXY` object is provided that serves as the proxy. The EPCF calls within the *IFrame* are delegated by the proxy layer to the upper EPCF layer. So instead of `EPCM` calls you use `EPCMPROXY`. The `EPCMPROXY` object is defined in the JavaScript file `epcfproxy.js` that comes with the portal. The JavaScript file `epcfproxy.js` has to be included into your portal application.

Usage

```
<HTML>
<HEAD>
<TITLE>ECPMPProxy test example</TITLE>
<!--
  This is a general proxy to delegate all EPCM calls to the upper fram
e -->
<SCRIPT src="epcfproxy.js"></SCRIPT>
<SCRIPT>//
  var lnDotPos = document.domain.indexOf( "." );
  if(lnDotPos>=0)document.domain = document.domain.substr( lnDotPos +
  1 );

  function run() {
    // call EPCF method via proxy (transparent for End-User
    EPCMPROXY.subscribeEvent( "urn:com.sapportals.portal.epcmdemo.anim
als",
      "onAnimalSelect", window, "handleEvent");
    showCurrentAnimal();
  }
```

Creating and Managing Content

```

function showCurrentAnimal() {
    var lsAnimal =
    EPCMPROXY.loadClientData("urn:com.sapportals.portal.epcmdemo.anima
ls",
                            "animalstored" );
    if (lsAnimal == null){ lsAnimal = "unknown"; }
    document.getElementById( "infoBox" ).innerHTML = lsAnimal;
}

function handleEvent( evt ) {
    showCurrentAnimal();
}

function showEPCMdata(){
    var data = ""
    data+= "\n EPCMPROXY.getUAType="+EPCMPROXY.getUAType()</STRONG>;
    data+= "\n EPCMPROXY.getUAVersion="+EPCMPROXY.getUAVersion()</STRO
NG>;
    data+= "\n EPCMPROXY.getUAPlatform="+EPCMPROXY.getUAPlatform()</ST
RONG>;
    data+= "\n EPCMPROXY.getUAVersion="+EPCMPROXY.getVersion()</STRONG
>;
    data+= "\n EPCMPROXY.getInstanceId="+EPCMPROXY.getInstanceId()</ST
RONG>;
    data+= "\n EPCMPROXY.getUniqueWindowId="+EPCMPROXY.getUniqueWindow
Id()
        </STRONG>;
    alert(data);
}
</SCRIPT>

</HEAD>
<BODY onLoad="run()">
<DIV class="header"> ECPMPProxy test component </DIV>
<P>
<BUTTONonClick="location.reload()">reload</BUTTON>
<BUTTONonClick="showEPCMdata()">show EPCM Data</BUTTON>
<DIVid="infoBox"></DIV>
</BODY>
</HTML>

```



There must be *document - domain* alignment so that the parent EPCM object can be accessed across the *IFrame* border. See section [JavaScript Origin Policy \[Page 33\]](#) for more details.

Use the *object-call-signature*, with the reference to the window object for the `EPCMPROXY.subscribeEvent()` method.

Restrictions

Restrictions for the *EPCFPROXY* object.

- Following APIs are available for the *EPCMPROXY* object:
 - [System API \[Page 33\]](#)

Creating and Managing Content

- [Event API \[Page 33\]](#)
- [Client data Bag API \[Page 33\]](#)
- [WorkProtect and Cross Navigation API \[Page 33\]](#)
- The *EPCMPROXY* object delegates the calls up **one** level. If your portal application uses additional *Framesets* or *IFrames*, the calls inside the *subframes* are not processed.
- To avoid the JavaScript errors, encapsulate the *EPCFPROXY* calls with JavaScript try/catch statements.
- The JavaScript file `epcfproxy.js` is not a part of the portal core. The file must be stored in the application code repository and delivered with the portal application.

3.1.4.3 EPCF Configuration

To configure the EPCF service you have to be logged in as administrator.

Configuration steps:

1. Choose the command *System Administration* in the top level navigation.
2. Choose the commands *System Configuration* → *Service Configuration*
3. Choose the *Browse* tag and open the node *Applications*.
4. Open the node *com.sap.portal.epcf.loader*.
5. Open the subnode *Services* and you will see the entry *epcfloader*.
6. Click the *epcfloader* entry with the right mouse key and select *Edit*.

The property page is displayed and you can modify the values. To save the changes, choose the *Save* button.



If you are working in a cluster environment, you have to restart the EPCF service so that the changes take immediate effect on all cluster nodes.

If the EPCF property values are not set correctly, the EPCF service uses the default settings at runtime.

EPCF Properties

Property	Description
<code>applet.archive = < on off ></code>	<p>Defines if the classes of the Java applet are transferred as single class files or in one Java archive (JAR). This setting has only an affect if the <code>framework.level</code> property is set to 2.</p> <p>on: All classes are loaded from the server to the client in one JAR file.</p> <p>off: Every class is loaded from the server to the client individually.</p> <p>For a productive system we recommend the value on.</p>

<pre>applet.trace.level = < 0 1 2 ></pre>	<p>This setting only has an affect if the <code>framework.level</code> property is set to 2.</p> <p>The <code>applet.trace.level</code> controls the level of error messages displayed. A higher <code>applet.trace.level</code> reduces the portal performance.</p> <p>0: Display errors only.</p> <p>1: Display errors and warnings.</p> <p>2: Display errors, warnings and information.</p> <p>For a productive system we recommend the value 0.</p>
<pre>framework.level = < 0 1 2 ></pre>	<p>Defines the EPCF service level in use. Please refer to section EPCF Level [Page 33] for more details.</p> <p>The default setting is 2.</p>

See also

[Work Protect Mode for EP 6.0 \[Page 33\]](#)

[Properties for EP 5.0 \[Page 33\]](#)

3.1.4.4 WorkProtect Feature for EP 6.0

To match the concept of the *WorkProtect* feature, a portal application must meet the following requirements:

- Maintain the *dirty indicator*
- Adjust portal links (This function is currently only supported by CRM).

Maintaining the Dirty Indicator

The *dirty indicator* status of a portal application informs the portal that there is unsaved data.



The portal application sets the dirty indicator when the user enters a new value into an input field. The portal application resets the dirty indicator when the user saves the value (for example when the user chooses the Save button).

See section [WorkProtect and Cross Navigation API \[Page 33\]](#) for more details.

Adjusting Portal Links

The portal can only check the current dirty indicator and perform the navigation without losing data, if the portal application replaces all the links that can destroy the contents of the content area with links having the following syntax (analogous to New Navigation Model / WorkProtect Mode , section Cross Navigation):

```
<A HREF=myLink onclick=return EPCM.doNavigate('any_PCD_URL')>
```

The parameter `<any_PCD_URL>` specifies the location of a page or an external service in the user role. Constants must be enclosed in quotation marks.

You can find the correct value for the page in the *Role Editor*.

Creating and Managing Content

Make sure that you update the corresponding parameter values for the `<PCD_URL>` in the secondary links and navigation targets when you change the role structure.

Configuration Test

Test tool: `com.sap.portal.epcf.loader.Dirty`

The test tool supports the tracing and solving of problems related to the *dirty indicator*. You can find the tool under

System Administration → Support → Support Desk → Client Framework

3.1.4.5 Navigation

The portal navigation model supports the *navigation* and *WorkProtect* feature. These features allow tight integration of *stateful* applications in the portal environment and improve the usability of the applications running in the portal.

See also:

[Navigation Service \[External\]](#)

[Navigation API \[Page 33\]](#)

[WorkProtect API \[Page 33\]](#)

Availability

The JavaScript based API has been introduced in EP 5.0.4.1 (see **SAP Note 543274**) and is compatible to the API in EP 6.0.

Business Case for Navigation

Portal implementations usually have separate areas for handling navigation and displaying content. The navigation area typically visualizes a navigation tree and highlights the selected node. The content area visualizes this selected node, for example, a portal page, document or portal application). This model resembles a simple file system browser and works well with *stateless* portal applications.

However, this approach does not meet all requirements for an enterprise solution, which should be able to handle complex business processes in parallel and switch from one context to another.

Navigation Target

The navigation target specifies the location of an iView or a page in the current user role. The target can be obtained from the portal catalog as a value that is concatenated by folder id s, roles or other objects.

The navigation target has to be prefixed with the corresponding navigation connector name that is used for retrieving the navigation structure. When accessing iViews and pages in the role from the Portal Content Directory (PCD), you have to add the prefix `ROLES://` to the URL.

Example:

We have created a custom role (`MyRole`) and assigned an iView (`MyIView`) to it:

```
portal_content (root folder) → MyRole (folder) → MyRole (role) →  
MyTest (folder) → MyIView (iView)
```

Creating and Managing Content

The corresponding navigation target is:

```
ROLES://portal_content/MyRole/MyRole/MyTest/MyIView
```

When you change the role structure, you have to update the corresponding values used in secondary links or used as navigation target; always keep both in sync.

Compatibility to EP 5.0

In the Enterprise Portal 5.0 the navigation target to the portal pages or external services are specified without the prefix for example, `/roles/MyEP50Role/MyEP50Folder/MyEPApp`. These navigation targets are also valid in EP 6.0.

When you have migrated the content from EP5.0 to EP6.0, the migrated content is in folder `portal_content/com.sap.portal.migrated/ep_5.0`. The portal navigation will check for incomplete navigation targets and it to the new schema automatically.

```
ROLES://portal_content/com.sap.portal.migrated/ep_5.0/roles/MyEP50Role/MyEP50Folder/MyEPApp
```

Navigation Features for Navigation Targets

The Enterprise Portal offers the following features for navigation targets:

- Start with Navigation Target

This feature lets you start the portal and automatically navigate to any portal page or iView inside the role.

- Navigation

This feature allows seamless *navigation* from portal applications. The primary links in the navigation as well as the secondary links in the content area can be used in a portal application. The navigation updates the content area correctly and highlights the corresponding node in the navigation tree - for a primary link or another instance.

Start with Navigation Target

The portal can be started in the browser with an URL that starts the first page assigned in the user role. The URL has the following structure:

```
<your_portal_server>/<portal_alias>
```

This URL can be extended by the navigation target that will be called automatically after the start. You have to specify a valid page (by default, `index.htm`). The extended URL has the following structure:

```
<your_portal_server>/<portal_alias>/<initial_page>?
NavigationTarget=<escaped_NavigationTarget>
```

The `<escaped_NavigationTarget>` parameter represents the *escaped* location of the page or iView in the role. *Escape* is necessary to avoid conflicts when using special characters. See the JavaScript function `escape` for more details.

Example:

```
http://myportal.wdf.sap.com:8100/irj/index.htm ?
NavigationTarget=ROLES%3A//portal_content/MyRole/MyRole/MyTest/MyIView
```

Navigation

The solution implemented for secondary links in the portal component uses a combination of a HTML hyperlink and a call of a EPCF service method [EPCM.doNavigate\(\) \[Page 33\]](#).

```
<A HREF="myLink" onclick="return EPCM.doNavigate('target') ">  
This is HTML Link</A>
```

The *String* parameter `target` represents the location of an iView or a page in the role. The value is available from the portal content catalog.



Constant *String* values must be enclosed in JavaScript in single quotes.

When the link is activate, the `EPCM.doNavigate()` method informs the Top Level Navigation (TLN) about the required navigation target. The TLN will handle this request it in the same way as any other navigation using primary links.

3.1.4.6 Glossary

Terms used in the EPCF service description.

3.1.4.6.1 Client Data Bag

The *client data bag* is a transient data buffer for the Web client (browser) which is active as long as the session of the browser. For the portal that is as long as the user is logged on to the portal.

An iView can access the *client data bag* with the *Enterprise Portal Client Manager* (EPCM) object.

3.1.4.6.2 JavaScript Origin Policy

The *JavaScript Origin Policy* controls the access to the Document Object Model (DOM) from different frames. Scripting between two frames is permitted only if both frame sources come from the same top level domain.

All browsers support the *JavaScript Origin Policy*, so foreign web sites are unable to retrieve data from the portal page or the iViews.

An similar origin policy also applies for the *Java Virtual Machine* (JVM). Classes/objects can only interact with classes/objects which are loaded from the same location. Therefore it is impossible for a foreign applet to access the data inside the Client Data Bag or use the Client Data Channel.

For more information see Microsoft documentation at internet address

msdn.microsoft.com/library/default.asp?url=/workshop/author/om/xframe_scripting_security.asp

and modzilla documentation at internet adress

www.mozilla.org/projects/security/components/same-origin.html

3.1.4.6.3 Namespaces

The World Wide Web Consortium (W3C) (<http://www.w3c.org>) defined the naming and addressing standards for Web development. The Enterprise Portal uses these standards in the EPCF service for the events and [Client Data Bag \[Page 33\]](#).

Uniform Resource Identifier (URI)

This section refers to *Request for Comments* (RFC). The comment for the specified RFC number can be found at <http://www.ietf.org/rfc/>.

Name Syntax

Some methods, for example, `EPCM.raiseEvent(...)`, expect a *Name* as argument. *Name* is a *String* variable with restricted characters.

Valid characters for *Name* are:

Range	Characters
Lowercase characters	a to z
Uppercase characters	A to Z
Numerical characters	0 to 9
Additional characters	Underscore(_), Dash (-)

Namespace Syntax

The namespace definition is compliant with the Unified Resource Name (URN) specification, which is available from the World Wide Web Consortium. Namespaces used in JavaScript functions calls must be compliant to this specification.

Namespaces reserved by the Enterprise Portal

Reserved name-space	Used for
com.sapportals.portal.*	Portal core development
com.sapportals.*	Portal core development

The namespace must start with the string "urn:" followed by the structure (in *Backus-Naur* form)

```
<URN>::="urn:" <Namespace_identifier> ":"
<Namespace_Specific_String>
```

The tokens `<namespace_identifier>` and `<Namespace_Specific_String>` must be compliant with the recommendation RFC 2141 and RFC 1630. We recommend that you use only lowercase, uppercase and numerical characters.

3.2 Uniform Resource Identifier (URI)

It addresses a resource in the Internet in the following way:

- By name
This is called *Uniform Resource Name* (URN).
- By location,
This is called *Uniform Resource Locator* (URL).

3.3 Uniform Resource Locator (URL)

It addresses a resource in the Internet. The URL is the address you enter into the address field of your browser. The URL syntax describes a subset of the *Uniform Resource Identifier* syntax.

3.4 Uniform Resource Name (URN)

It addresses a resource in the Internet, regardless of its location. The URN syntax follows the rules of the URI. A URN can also be used to define distinct entities without being associated to an existing resource. The name spaces in the portal make use of this feature. A URN has the prefix: `urn://`. For further information about the syntax, see the description for *RFC 2141* at www.ietf.org.

3.4.1 Page Builder

Purpose

The *page builder* component loads the portal applications and builds the page so that it is displayed properly in the browser. The page building process has two stages:

1. The *page builder* gets the list of portal applications that are on that page from the page editor. It reads the properties of the portal applications and loads the portal applications using the Portal Runtime (PRT).
2. The *page builder* loads the *page layout* component that is based on a JSP file. It represents the actual visual layout of the page and can be edited in the *layout view* of the *page editor*. The *page layout* component takes the response from every portal application, wraps it with a tray and places it on the page at the right location.

The page building process is done asynchronously, therefore the loading time of the page takes not longer than the time it takes for the slowest portal application in the page to load (unless a specific Page Timeout was defined).

Page Layout Component

The *page builder* layout is defined by JSP templates with a dedicated tag library for organizing portal applications on a page. The main tag of the tag library is the *container tag* `<lyt:container>` which serves as space holder for *columns* (and *rows* in a future release) of portal applications. Every container turns into a HTML table with a portal application in every cell.

If a portal application should be placed in a *tray*, the *page layout* wraps it with an HTML Business for Java (HTMLB) *tray* control.

3.4.1.1 Isolation Modes

EP6.0 *page builder* offers the following isolation modes for a portal application:

- EMBEDDED
- URL

See also

[Enterprise Portal Client Framework \(EPCF\) \[Page 33\]](#): Client events.

EMBEDDED

The generated content for the portal application is embedded into the HTML code of the page without transformation. It is part of the page content. The *page builder* has no control over the appearance of the content.

The EMBEDDED isolation mode has following features:

- The content of the portal applications is generated on the server without additional requests from the browser to the server.

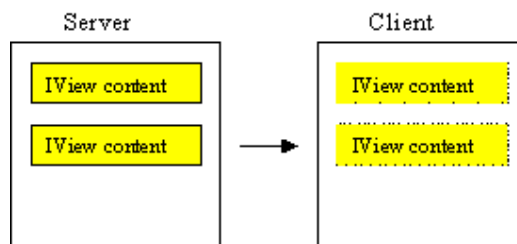
Uniform Resource Name (URN)

- All portal applications on the page are called asynchronous on the server and sent in one page response to the client.
- Interactions with the portal application and *reload* actions will reload the entire page. This results in flickering pages and lowers the performance, when all portal applications are reloaded.
- The portal application window size is not controlled by the page. The window height parameter has no effect on the portal application when in EMBEDDED mode.
- The *page builder* does not add scroll bars to the window of the portal application.
- Server (PRT) events and client (EPCF) events are supported.

Limitations

- HTMLB portal applications, which have to keep the state of input fields by form, will lose the data of the input fields when other portal application forms on the page are submitted.
- Request parameters to keep the portal application state, will be lost when other portal applications also keep the state.
- The portal application is part of the entire HTML code of page. Therefore the script names of a portal application have to be unique, to avoid collisions with other scripts of other portal applications on the page.
- The HTMLB *ScrollContainer* control with the height attribute set to 100% will shrink to zero height.
- External content (to the portal server) is very limited and must be handled with special care (for example, resources and cookie handling). *URLViewsRuntime* can not be used.
- Portal application can only use the same codepage (charset) for the page (UTF-8).

Schematic



URL

The portal application is in an *IFrame*, isolated from the HTML code of the page.

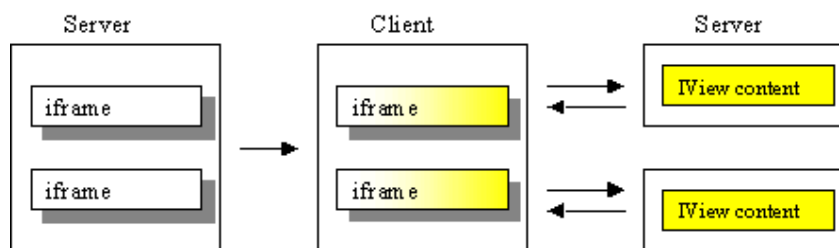
The URL isolation mode has following features:

- The Page Builder generates the *IFrame* and sets the source attribute (SRC) to point to the URI of the portal application.
- The content of the portal application comes in an additional server request.

Uniform Resource Name (URN)

- Interactions with the portal application and *reload* actions will affect only the portal application itself and not the entire page. The result is a flicker-free page and a performance advantage since not all portal applications are reloaded.
- The *page builder* has full control over the *IFrame* size. The *page builder* can set portal application window height to a *fixed*, *full page* or *automatic* according to the content of the portal application.
- The *IFrame* is generated with scroll bars.
- The width of portal application window can be controlled better, since the *IFrame* has a horizontal scroll bar when the content exceeds the window width.
- Only client events (EPCF) are supported.

Schematic



Recommended Usage of Isolation Modes

If your content is large and the client rendering takes too long, switch to URL or EMBEDDED isolation mode.

Use the EMBEDDED isolation mode when the portal applications on the page are connected with Portal Object Model (POM) events to other portal applications and need to *repaint* themselves, through the server, according to client actions.

Use the EMBEDDED isolation mode also for pages (as the page property) and for portal applications that have no other portal application on the page (as a navigation node).

Use the URL Isolation Mode when it is required to present external content (not from the portal server) or several portal applications on a page that need different codepages (charsets).

It is strongly recommended not to use EMBEDDED isolation mode and URL isolation mode on same page.

Portal pages should have EMBEDDED isolation mode to save an additional request to fetch the page.

External content should have URL isolation mode.



Test your portal application always in the portal environment. Execute it on a page in the portal framework with the other portal applications on the page.

3.4.1.2 Page Builder API

The *page builder* API provides methods to control the page size, refresh the page and page personalization.

Uniform Resource Name (URN)

In this chapter we refer to term `iView ID`. The `iView ID` is the URI of the `iView` in the Portal Content Directory (PCD).



Full specified URI on an `iView`:

```
pcd:portal_content/every_user/general/eu_role/com.sap.portal.my
pages/com.sap.portal.eu_ws/com.sap.portal.home/com.sap.portal.
pageBuilderDefault
```



The `iView` domain has to be *relaxed* before using the *page builder* API.

For native portal components, Java `iViews`, this is done automatically by the EPCM object. See section [JavaScript Origin Policy \[Page 33\]](#) for more details.

See also

[Isolation modes \[Page 33\]](#)

3.4.1.2.1 Client API

The methods in this section are provided by the JavaScript object *pageSupport* that exists on every portal page. Although the *pageSupport* object resides on the page document, `iViews` using the URL isolation mode must refer to their parent document to get the *pageSupport* object.

pageSupport.getIvuId (wndRef)

This method returns the `iView ID` of the `iView` specified by parameter `wndRef` as type *String*.



An alternative to this method, is to use PRT API on the server.

```
request.getComponentContext ().GetContextName ()
```

Use the client-side method if your `iView` uses URL isolation mode and if you have no access to the PRT API.

Parameter Description

Parameter	Type	Description
<code>wndRef</code>	Object	The HTML window object of the <code>iView</code> .

Usage

```
<SCRIPT language ="JavaScript">
    var myId = parent.pageSupport.getIvuId(self);
</SCRIPT>
```

pageSupport.getIvuFrameObj (ivuld)

This method returns the iView *IFrame* object of the iView specified by the iView ID as type *Object*.

This method is only relevant for iViews using URL isolation mode.

Parameter Description

Parameter	Type	Description
ivuld	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">
    ...
    var myId = parent.pageSupport.getIvuId(self);
    var myIframe = parent.pageSupport.getIvuFrameObj(myId);
    myIframe.style.width += 100;
    ...
</SCRIPT>
```

pageSupport.getIvuWindow (ivuld)

This method returns the iView window of the iView specified by the iView ID as type *Object*.

This method is only relevant for iViews using URL isolation mode.

Parameter Description

Parameter	Type	Description
ivuld	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">
    ...
    var myId = parent.pageSupport.getIvuId(self);
    var myWindow = parent.pageSupport.getIvuFrameWindow(myId);
    myWindow.location.href = "url";
    ...
</SCRIPT>
```

pageSupport.ivuExpand (ivuld)

This method opens an iView, specified by the iView ID, in a new window. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

This method simulates the *Open in new window* option of the iView *tray*.

Parameter Description

Parameter	Type	Description
ivuId	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">
...
var myId = parent.pageSupport.getIvuId(self);
parent.pageSupport.ivuExpand(myId);
...
</SCRIPT>
```

pageSupport.ivuRefresh (ivuId)

This method refreshes an iView specified by the iView ID. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

If the iView content is in the portal cache it will be replaced by new content generated for this iView.

This method simulates the *Refresh* option of the iView *tray*.

Parameter Description

Parameter	Type	Description
ivuId	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">
...
var myId = parent.pageSupport.getIvuId(self);
parent.pageSupport.ivuRefresh(myId);
...
</SCRIPT>
```

pageSupport.ivuReload (ivuId)

This method reloads an iView specified by the iView ID. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

The iView content is retrieved from the portal cache. See method `pageSupport.ivuRefresh(ivuID)`.

Parameter Description

Parameter	Type	Description
-----------	------	-------------

Uniform Resource Name (URN)

ivuId	String	iView ID of the iView.
-------	--------	------------------------

Usage

```
<SCRIPT language ="JavaScript">
...
var myId = parent.pageSupport.getIvuId(self);
parent.pageSupport.ivuReload(myId);
...
<SCRIPT>
```

pageSupport.ivuPersonalize (ivuId)

This method opens the personalization dialog for the iView specified by the iView ID. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

This method simulates the *Personalize* option of the iView *tray*.

Parameter Description

Parameter	Type	Description
ivuId	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">
...
var myId = parent.pageSupport.getIvuId(self);
parent.pageSupport.ivuPersonalize(myId);
...
<SCRIPT>
```

pageSupport.ivuHelp (ivuId)

This method opens the help component for the iView specified by the iView ID. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

This method simulates the *Help* option of the iView *tray*.

Parameter Description

Parameter	Type	Description
ivuId	String	iView ID of the iView.

Uniform Resource Name (URN)

Usage

```
<SCRIPT language ="JavaScript">

...

var myId = parent.pageSupport.getIvuId(self);

parent.pageSupport.ivuHelp(myId);

...

</SCRIPT>
```

pageSupport.ivuRemove (ivuld)

This method removes the iView, specified by the iView ID, from the page. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

When the iView was added to the page by the user with the personalization dialog, the iView will be deleted from the page.

When the iView was set on the page by the administrator, the iView will be hidden and not deleted. The iView can be made visible again with the personalization dialog by the user.

This method simulates the *Remove from page* option of the iView *tray*.

Parameter Description

Parameter	Type	Description
ivuld	String	iView ID of the iView.

Usage

```
<SCRIPT language ="JavaScript">

...

var myId = parent.pageSupport.getIvuId(self);

parent.pageSupport.ivuRemove(myId);

...

</SCRIPT>
```

pageSupport.ivuAdjustHeight (ivuld [,height])

This method adjusts the height of the *IFrame* that contains the iView, specified by the iView ID. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

This method is only relevant for iViews using URL isolation mode.

Use this method when the iView changes its size and you want the *IFrame* to change its size also. If you do not specify the *height* parameter, the *IFrame* height will be adjusted to the current size of the iView. This is same behaviour as for iViews that have the *automatic* height property.

Parameter Description

Uniform Resource Name (URN)

Parameter	Type	Description
ivuId	String	iView ID of the iView.
Height (optional)	Integer	Height of the <i>I</i> Frame.

Usage

```

<SCRIPT language ="JavaScript">
    ...
    var myId = parent.pageSupport.getIvuId(self);
    parent.pageSupport.adjustHeight(myId, 500);
    ...
</SCRIPT>

```

pageSupport.ivuRecalcTray (ivuId)

This method is for Netscape browsers to adjust the *tray* size (which contains the iView) according to iView content. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

Parameter Description

Parameter	Type	Description
ivuId	String	iView ID of the iView.

Usage

```

<SCRIPT language ="JavaScript">
    ...
    var myId = parent.pageSupport.getIvuId(self);
    parent.pageSupport.ivuRecalcTray (myId);
    ...
</SCRIPT>

```

pageSupport.ivuAddTrayOption (ivuId, optionTitle, optionFunct)

This method add a new entry to the *tray* (which contains the iView) options menu. The method returns a *boolean* value that is *false* if the method failed and *true* if the method was successful.

For multilingual application, the iView developer has to take care about the translation of the *optionTitle* parameter.

Parameter Description

Parameter	Type	Description
-----------	------	-------------

Uniform Resource Name (URN)

ivuId	String	iView ID of the iView.
optionTitle	String	Title of the option. The title is displayed in the option menu of the tray.
optionFunc	String	JavaScript code that will be executed when the option is chosen.

Usage

```
<SCRIPT language ="JavaScript">

    ...

    var myId = parent.pageSupport.getIvuId(self);

    parent.pageSupport.ivuAddTrayOption(myId,"hello","alert('hello')");

    ...

</SCRIPT>
```

3.4.1.2.2 Client Events

The events in this section are provided by the JavaScript object EPCM that exists on every portal page, when the [EPCF \[Page 33\]](#) service is activated.

External portal applications that have no EPCM object must use the regular page client API.

The events expect a parameter in the form:

```
{Id : "iview id" , Window : iview HTML frame window}
```

If the iView ID is not *null*, the *window* parameter is ignored. The *window* parameter was designed to support Java portal applications and external iViews using the URL isolation mode. According to the kind of iView set the parameter as follows:

- For a Java portal application, specify the iView ID and set the *window* parameter to *null*.
- For an external iView, set the iView ID to null and the *window* parameter to *self*.

You get the iView ID using the Portal Runtime (PRT) API on the server with following command:

```
request.getComponentContext ().GetContextName ();
```

See also

[Isolation modes \[Page 33\]](#)

[Client API \[Page 33\]](#)

[EPCF service \[Page 33\]](#)

The page builder has following events:

Uniform Resource Name (URN)

expand

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "expand", {Id:id, Window:window})

The *expand* event opens an iView in a new window.

This event simulates the *Open in new window option* of the iView tray.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```
<SCRIPT language ="JavaScript">
    ...
    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "expand",
        {Id:"<%=componentRequest.getComponentContext().GetContextName() %>"
        , Window:null})
    ...
    // For external iviews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "expand",
        {Id:null, Window:self})
    ...
</SCRIPT>
```

refresh

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "refresh", {Id:id, Window:window})

The *refresh* event refreshes an iView. If the iView content is in the portal cache it will be replaced by new content generated for this iView.

This event simulates the *Refresh* option of the iView tray.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Uniform Resource Name (URN)

Usage

```

<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "refresh",

    {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
    Window:null})

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "refresh",
    {Id:null, Window:self})

    ...
</SCRIPT>

```

reload

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "reload", {Id:id, Window:window})

The *reload* event reloads an iView. The iView content is retrieved from the portal cache.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```

<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "reload",

    {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
    Window:null})

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "reload",
    {Id:null, Window:self})

    ...

```

```
<SCRIPT>
```

personalize

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "personalize", {Id:id, Window:window})

The *reload* event starts the personalization dialog. This event simulates the *Personalize* option of the iView tray.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```
<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:

    EPCM.raiseEvent("urn:com.sapportals:pagebuilder",
"personalize",

    {Id:"<%=componentRequest.getComponentContext().GetContextName() %
>",
    Window:null})

    // For external iViews

    EPCM.raiseEvent("urn:com.sapportals:pagebuilder",
"personalize",
    {Id:null, Window:self})

    ...

<SCRIPT>
```

remove

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "remove", {Id:id, Window:window})

The *remove* event removes the iView from the page.

When the iView was added to the page by the user with the personalization dialog, the iView will be deleted from the page.

When the iView was set on the page by the administrator, the iView will be hidden and not deleted. The iView can be made visible again with the personalization dialog by the user.

This event simulates the *Remove from page* option of the iView tray.

Uniform Resource Name (URN)

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```
<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "remove",

    {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
      Window:null})

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "remove",
      {Id:null, Window:self})

    ...

</SCRIPT>
```

*help***EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "help", {Id:id, Window:window})**

The *help* event starts the help component for the iView. This event simulates the *Help* option of the iView tray.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```
<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "help",

    {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
      Window:null})

    ...

</SCRIPT>
```

Uniform Resource Name (URN)

```

        Window:null}}

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "help",
        {Id:null, Window:self})

    ...
<SCRIPT>

```

adjustHeight

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "adjustHeight", {Id:id, Window>window, [,Height:height]})

The *adjustHeight* event adjusts the height of the *IFrame* that contains the *iView*.

This method is only relevant for *iViews* using URL isolation mode.

Use this event when the *iView* changes its size and you want the *IFrame* to change its size also. If you do not specify the height parameter, the *IFrame* height will be adjusted to the current size of the *iView*. This is same behaviour as for *iViews* that have the automatic height property.

Parameter Description

Parameter	Type	Description
id	String	<i>iView</i> ID of the <i>iView</i> .
Window	Object	The HTML window object of the <i>iView</i> .
height	Integer	Height of the <i>IFrame</i> .

Usage

```

<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder",
        "adjustHeight",

        {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
        Window:null})

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "
        adjustHeight ",
        {Id:null, Window:self})

    ...
<SCRIPT>

```

Uniform Resource Name (URN)

recalcTray

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "recalcTray", {Id:id, Window:window})

This event is for Netscape browsers to adjust the *tray* size (which contains the iView) according to iView content.

Parameter Description

Parameter	Type	Description
id	String	iView ID of the iView.
Window	Object	The HTML window object of the iView.

Usage

```
<SCRIPT language = "JavaScript">
    ...
    // For Java portal components:
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder",
        "recalcTray",
        {Id:"<%=componentRequest.getComponentContext().GetContextName() %>",
        Window:null})

    // For external iViews
    EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "recalcTray",
        {Id:null, Window:self})
    ...
</SCRIPT>
```

addOption

EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "addOption", {Id:id, Window:window, Caption:caption, Func:func})

The *addOption* event adds a new entry to the *tray* (which contains the iView) options menu.

For multilingual application, the iView developer has to take care about the translation of the *caption* parameter.

Parameter Description

Parameter	Type	Description
Id	String	iView ID of the iView.

Uniform Resource Name (URN)

Window	Object	The HTML window object of the iView.
Caption	String	Title of the option. The title is displayed in the option menu of the tray
Func	String	JavaScript code that will be executed when the option is chosen.

Usage

```

<SCRIPT language ="JavaScript">

    ...

    // For Java portal components:

        EPCM.raiseEvent("urn:com.sapportals:pagebuilder", "addOption",

        {Id:"<%=componentRequest.getComponentContext().GetContextName()%>",
        Window:null, Caption:"New Option", Func:"alert('new option')"}))

    // For external iViews

        EPCM.raiseEvent("urn:com.sapportals:pagebuilder", " addOption
        ",
        {Id:null, Window:self, Caption:"New Option",
        Func:"alert('new option')"}))

    ...

</SCRIPT>

```

3.4.1.3 Advanced Features

Hide Portal Application

A portal application can notify the *page builder*, at runtime, that it should not be included in the response. This feature can be utilized for many scenarios when a certain portal application is located on a page should not be displayed, according to its logic. The *detailed navigation* iView uses this technique to hide itself when no relevant navigation tree can be displayed.

To use this feature, the portal application must execute the following code in the `doOnNodeReady()` method:

```
request.getNode.setNodeMode(new NodeMode("HideMode"))
```



This feature is only available for portal applications using the [EMBEDDED \[Page 33\]](#) isolation mode.

Redirect Portal Application

A portal application using the [URL \[Page 33\]](#) isolation mode can instruct the *page builder* to reference a URL from the generated *IFrame* named `src`, instead of including the portal

Uniform Resource Name (URN)

application itself. Therefore, a portal application can be used to generate a URL and launch it from the page.

There are following requirements to use this feature:

- Include the following property to the `<component config>` section in the property file (`portalapp.xml`) of the portal application:

```
<property name="com.sap.portal.reserved.iframe.Redirect"
  value= "true"/>
```

- Portal application must use the URL isolation mode.
- Use the following code in the `doOnNodeReady(..)` method:

```
request.redirect (NewURL) ;
```

New Connection for Portal Application

A portal application using the [URL \[Page 33\]](#) isolation mode can use another connection than the portal Runtime (PRT) default connection

(`com.sapportals.portal.prt.connection`). This avoids the PRT embedded handling of certain document hooks, HTML tags handling, for example, `<head>`, `<body>` and so on.

To implement new connection you have to implement the *IPortalURLBuilder* interface and register it in the portal registry under `/runtime/urlbuilders/<NEW CONNECTION POINT>`.

The property file of the portal application (`portalapp.xml`) needs a property entry in the `<component config>` section so that the *page builder* is notified to use the new implemented connection. The property entry has the following format:

```
<property name="RequiredConnectionPoint" value="<NEW
CONNECTION>" />
```

JavaScript in PageLevel

A portal application can notify the page to add JavaScript files on the *page builder* document level. This feature is relevant for isolated portal application only.

To use this feature, the portal application has add the following properties to the property file (`portalapp.xml`) in the `<component-profile>` section:

```
<property name="com.sap.portal.reserved.iframe.PageLevelScript_1"
  value="<SCRIPT 1 ID>" >

  <property name="Path" value="<SCRIPT 1 PATH>" />
</property>

<property name="com.sap.portal.reserved.iframe.PageLevelScript_n"
  value="<SCRIPT N ID>" >

  <property name="Path" value="<SCRIPT NPATH>" />
</property>
```

Page Timeout

A Page Timeout property can be defined to set maximum (server) loading time for the page.

Uniform Resource Name (URN)

Usually the page will be sent to the browser as soon as all of the portal applications have been loaded. If a page that has set the *page timeout* property to 15 seconds contains an portal application that requires more than 15 seconds to load, a time out message instead of the portal application is displayed. Following scenarios are possible:

- If the portal application is cacheable, and this is the first time the page is loaded, the *page builder* continues to load the portal application in the background and if the page is refreshed after the portal application finished loading, the portal application content will be displayed in place from the cache.
- If the page was already loaded once and the portal application was already cached, the portal application will come from the cache. This is a fast process and there is no problem with *page timeout*. But if the cached content of the portal application is not valid, the portal application has to be reloaded from the site. If the portal application did not finish loading when the *page timeout* occurred, the page will display the old cached content of the portal application with a message indicating that it is the old cached content. The portal application is still loaded in the background so in the next time the page is reloaded it will display the correct cache content.

An administrator or a page developer must avoid the case where a non-cacheable, slow portal application, with EMBEDDED isolation mode, resides on a page with a *page timeout* lower than the actual portal application loading time. The result would be that the portal application is not loaded into the cache in the background and the portal application will never be displayed on the page.

A portal application that uses the URL isolation mode will load the portal application directly from browser, without considering the *page timeout* property, when the *Refresh* option has been chosen.

To the *page timeout* property add the following property to the property file (portalapp.xml):

```
com.sap.portal.page.PageTimeout=20000
```

The value is defined in milliseconds. The default value is 15 seconds = 15000 milliseconds. A *page timeout* value of -1 defines, that there is no time out and that the page will wait for all of the portal applications to finish loading as long as necessary.

Param List

Portal applications using URL isolation mode are loaded by the browser from an *IFrame*. The portal applications are completely isolated from the page structure and from the page load process and therefore are also isolated from the initial page request. If the initial page request contained parameters, these parameters will not be passed to the portal application using the URL isolation mode. To change this behavior, you have to add the property

`com.sap.portal.reserved.iview.ParamList` to the property file of the portal application (portalapp.xml). The property contains a list of parameters for the portal application. There are following options:

- The list contains the parameters for the portal application URL:

```
<property name=" ...ParamList" value="param1,param2,param3"/>
```
- To pass on all of the parameters from the request to the portal application you can use a wildcard (*):

```
<property name=" ...ParamList" value="*/>
```
- To pass on a subset of the parameters from the request to the portal application you can use a wildcard (*) and define the parameters you want to exclude:

Example:

```
<property name=" ...ParamList" value="*,param4,param5"/>
```

This will pass all of the request parameters excluding parameter 4 and 5.

3.4.2 HTML-Business for Java

Purpose

The HTML-Business for Java document is a supplement to HTML-Business for Java reference.

The document shows the usage of the controls in the JSP and how to call and process the JSP in the DynPage.

3.4.2.1 What is HTMLB?

HTMLB (HTML-Business for Java) provides a full set of easy-to-use Web controls. These guidelines describe the HTMLB controls, their types, usage, attributes, and how to set the attributes with the JSP-*taglib* and the *classlib*.

For each control there is a general page that describes its usage, types, and design-relevant attributes. A further page describes more technical issues, such as browser compatibility, editing possibility in the Style Editor, and accessibility issues. Thirdly, the Control API page provides a development-oriented view of the control with detailed descriptions of attributes and parameters.

In addition, there are pages that describe general interaction design aspects, such as page layout, correct usage of certain often-used controls, as well as hints on finding the right control for a given purpose.

Knowledge of Java, JSP (information can be found at internet address java.sun.com) and HTML (information can be found at internet address www.w3.org) is helpful when reading this document.

Basic Idea

HTMLB allows a design-oriented page layout. It is designed to overcome typical servlet problems, such as:

- Visualization and business logic are not separate.
- Content management consumes a lot of qualified manpower. Skills in HTML, CSS, JavaScript etc. are essential.
- Development has to take care of different web clients and -versions.
- Maintaining the corporate identity through out the whole application is hard to achieve.
- Namespace conflicts with form elements

HTMLB provides the technological infrastructure for easy customer branding. See [Customer Branding and Style Editor \[Page 33\]](#).

How it Works

HTML-Business for Java provides the user with an efficient set of controls - similar to Swing/AWT. The controls are based on servlets and JSP pages. The developer uses bean-like components or JSP tags. Renderer classes finally translate the components into HTML-commands.

To demonstrate the similarity from HTMLB to Swing/AWT some synonyms.

Uniform Resource Name (URN)

HTMLB	Swing/AWT
Form	ContentPane, JFrame, JDialog
ControlComponent	JComponent
Container	ContainerContainer
Event	AWTEvent, InputEvent ...

Form

It is basically the wrapping paper of your page and essential for the data transfer from the web client to the web browser and for the event handling. Controls in the form must have unique control names. The control names are generated by the HTML-Business for Java renderer - therefore you cannot use for example, JavaScript to manipulate the controls.

Controls

GUI elements that are used to build an application. The controls are placed in a form. Every control has different attributes that define the "look" of the control. Controls are checkboxes, radio buttons and grids to name a few.

Some HTMLB controls**Container**

Container contain controls. Containers can contain containers - nesting. A simple container would be a 'tray' containing a 'gridLayout'. The gridLayout contains 'textView' and 'inputField'.

Events

Components can respond to user action. The response is called an event. An event usually causes a submit (sending the form from the web client to the web server). With the control that can create an event you specify the name of the event handling routine. The web server receives the form, analyzes it and calls the event handling routine which does the further processing.

Mobile Features

The mobile features enhance the functionality of HTML-Business for Java for mobile devices such as Pocket PCs, WAP-enabled mobile phones and other mobile device/browser combinations. The mobile features support a device-independent development of components for mobile devices by providing special renderer classes. These renderer classes consider the special features of different mobile devices and the browsers used.

For more information, see [Mobile HTMLBusiness for Java \[Page 33\]](#) and [Mobile Extensions to the Java Servlet Containers \[Page 33\]](#).

3.4.2.2 About the Reference

Structure of the Description

The description of the controls is structured in:

- General description - what is it
- Attributes of the control
- Overview of the attributes with possible values, defaults and the manipulation with the JSP-taglib and classlib.
- Example

The **M** column in the overview table specifies if the parameter is mandatory. A mandatory parameter is marked with a * in that column.

The **values** column in the overview table specifies which type of parameter the attribute expects. Possible entries are:

- String
An ASCII string. Usually event handling routines, names, titles etc.
- String (cs)
A case sensitive ASCII string. Usually event handling routines, names, titles etc.
- Units
An integer value specified in web client units. According to the HTML standard units can be specified in:

- Pixel (px)

Pixels are the smallest addressable unit on the web client. A web client has a maximum resolution, that is the number of horizontal times vertical pixel (for example, 800x600, 1024x768 etc.) When you specify units in pixel you can make sure that your control is displayed on every web client in the same size.

Pixel is the default unit.



Both expressions set the width of a control to 500 pixel.
width="500"
width="500px"

- Percent (%)

Uniform Resource Name (URN)

The percentage specified is calculated from the visible space of the web client. If for example, a width of a control is specified with 50% the control uses half of the of the web client width. The control changes its width according to the web client dynamically (for example, if the web client window gets scaled).



The width of a control is set to 30% of the web client.
width="30%"

The default value for the attribute is marked with a value in parenthesis, if it applies, for example (100).

- Numeric

A numeric expression.

- Others

If an attribute requires specific values. Booleans require "TRUE" or "FALSE" or text size can only be "LARGE", "MEDIUM" and "SMALL". Default values are marked with (d).



When the default value for the text size is MEDIUM it is depicted as:

LARGE
MEDIUM (d)
SMALL

Controls

General

To use the controls you have to know about the syntax and the attributes of the controls. Every control has different attributes. In the description we describe the attributes and gather the information in a table which shows the usage with the taglib and the classlib.

Import Statements

When you use the methods in a Servlet, Abstract Portal Component or DynPage, you have to import the controls. The IDE, like Eclipse, give you a hint if an import statement is needed and suggest an import statement. A HTMLB control, for example inputField, is defined in two packages, so the IDE will offer you two choices:

```
com.sapportals.htmlb.InputField
com.sapportals.htmlb.unifiedrendering.controls.InputField
```



The `com.sapportals.htmlb.InputField` package is the one to use. The `com.sapportals.htmlb.unifiedrendering.controls.*` package is for internal use only.

Syntax

Programming with the JSP taglib follows the XML syntax. Each control is "wrapped" in tags. To identify the tags as XML the prefix

Uniform Resource Name (URN)

hbj: (stands for: HTML-Business for Java)

is used. Some controls (for example, tray, group) also need a tag body. The tag body specifies the controls that are placed "inside" the tag. The syntax would be like:

```

Tag
<hbj:control          comment: begin of tag for HTMLB control
      attributes      comment: setting of attributes of HTMLB control
</hbj:mycontrol>      comment: end of tag for HTMLB control

Tag with "quick" end of tag (only possible when the tag has no body)
<hbj:control          comment: begin of tag for HTMLB controls
      attributes      comment: setting of attributes of HTMLB control
/>                   comment: end of tag for HTMLB controls

Tag with body
<hbj:control          comment: begin of tag for HTMLB control
      attributes      comment: setting of attributes of HTMLB control
<                    comment: end of tag for HTMLB control
  <hbj:a_control_in_the_body
      attributes
  />
  <hbj:next_control_in_the_body
      attributes
  />
  more controls

</hbj:control>        comment: end of tag for HTMLB controls with body

```

Scriptlet

A scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Within a scriptlet, you can do any of the following:

- Declare variables or methods to use later in the file.
- Write expressions valid in the page scripting language.
- Use any of the implicit objects or any object declared with a `<jsp:useBean>` element.
- Write any other statement valid in the scripting language used in the JSP page (if you use the Java programming language, the statements must conform to the Java Language Specification).

Any text, HTML tags, or JSP elements must be written outside the scriptlet.

Scriptlets are executed at request time, when the JSP container processes the client request. If the scriptlet produces output, the output is stored in the out object.

Certain attributes (if the column "JSP taglib" in the attribute table to each control has no entry) can only be assigned using scriptlets. The scriptlet has to be placed in the tag body of the HTMLB control. The scriptlet starts with `<%` and ends with `%>`. The following example uses the [button \[Page 33\]](#) control and sets some attributes with a scriptlet.

Uniform Resource Name (URN)

Example

```

<hbj:button
  id="OrderConfirm"
  width="100px"
  tooltip="Click here to confirm order"
  onClick="ProcessConfirm"
  disabled="false"
  design="STANDARD"
>
<%      comment: start scriptlet in the tag body
// set the text for the button
OrderConfirm.setText("Confirm");
// set "width" - this overrides "width" set in attribute section
OrderConfirm.setWidth("125px");
%>      comment: end of scriptlet
</hbj:button>

```

Result**Enumeration Values**

In the classlib column some values have to be set as enumeration values. In the classlib column you find the class name and the `enum` (separated by a dot).



```
breadcrumb.setSize(BreadCrumbSize.MEDIUM)
```

For an executable program you have to add the location of the enum. That is:

```
com.sap.htmlb.enum.
```

So according to the example above you have to specify:

Your program:

```
breadcrumb.setSize(com.sap.htmlb.enum.BreadCrumbSize.MEDIUM);
```

To save some typing when you enumeration values more often the package can be imported:

```
<%@ page import="com.sap.htmlb.enum.BreadCrumbSize, ..... " %>
```

Boolean Values

Taglib:

Boolean values are specified as string and can be lowercase and/or uppercase.

Classlib:

Boolean values are specified as **boolean** and have to be specified only in lowercase characters.

3.4.2.3 General

Purpose

General information about the user interface, how to make customer branding, error handling and general accessibility information.

[Customer Branding and Style Editor \[Page 33\]](#)

[Error Handling](#)

[\[Page 33\]](#)[Accessibility of HTMLB Controls \[Page 33\]](#)

3.4.2.3.1 Customer Branding and Style Editor

Purpose

Portal software must reflect the customers corporate identity and branding guidelines. For this reason, we provide a technological infrastructure and tools to support customers in this goal. The current portal release offers a certain amount of design flexibility that allows our customers to fulfill their branding needs.

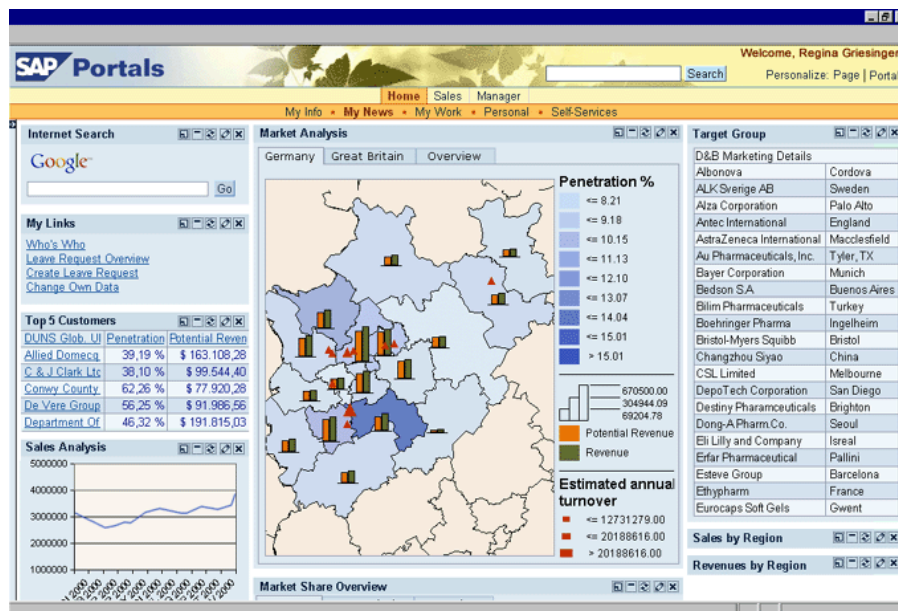
This flexibility is achieved by:

- Placing all design information into cascading style sheets (CSS) instead of writing it directly into the code. As far as possible, images are defined with the CSS attribute background-image.
- Using only central CSS in all HTMLB controls.
- Shipping predefined design variants (color templates) of the portal among which our customers can choose.
- Providing the Style Editor tool for supporting branding activities at the customers' sites.

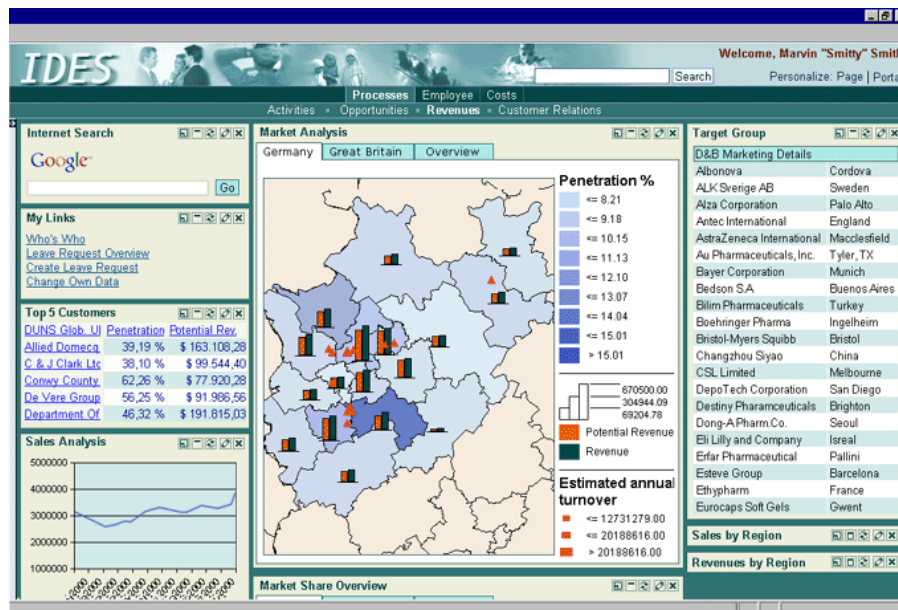
Below you see the portal with the *Mango and Polarwind* standard design and the same portal with a customized design.

Portal standard design Mango and Polarwind

Uniform Resource Name (URN)



Example of a customized design



Further Information

[Style Editor \[Page 33\]](#)

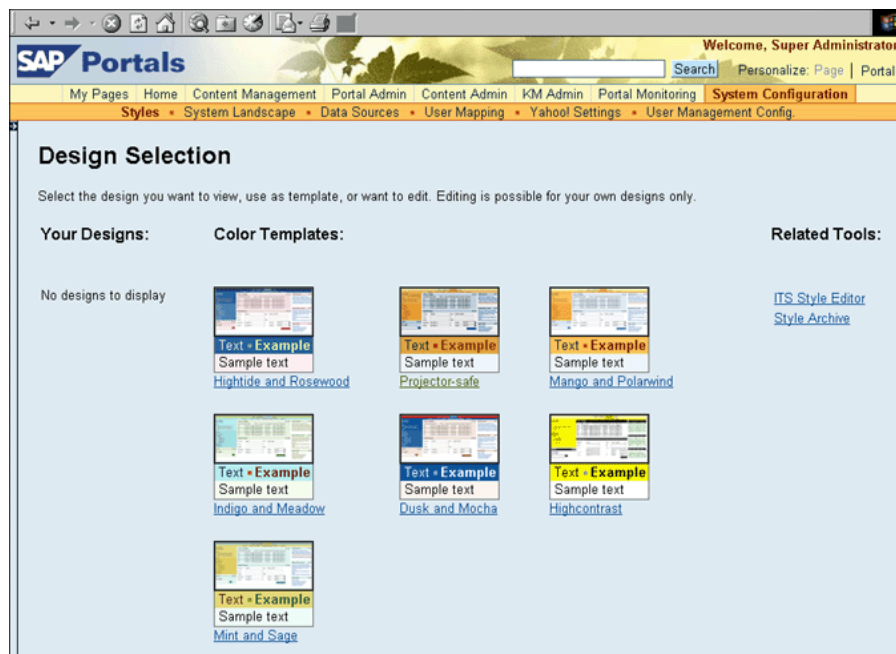
[HTMLB Controls and Style Editor \[Page 33\]](#)

3.4.2.3.1.1 Style Editor

Purpose

The Style Editor is a Web-based tool, which allows a designer or administrator to copy and then modify any of our predefined color templates to create a new design. With the Style Editor, an authorized user can change the look-and-feel of the portal without having to be an HTML expert. For example, no knowledge about CSS attribute names is required. Below, you see the entry screen of the Style Editor, where users can select between predefined and customized designs, provided the customer created such.

Design selection screen showing all available color templates

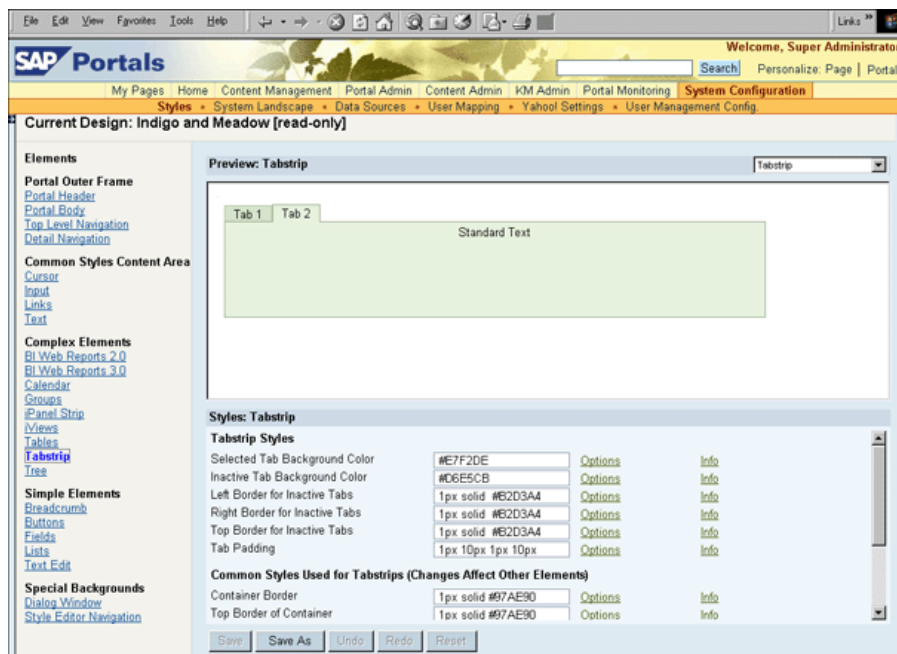


A clearly defined number of styles, such as the background colors, font colors, or images are presented on the user interface. Users can immediately check their changes in the preview area.

The Style Editor creates the CSS files for all platforms and browser versions that SAP Portals supports. Note that style sheets cannot be edited directly. The following example shows the user interface for changing the look of the tabstrip control.

Customizing the tabstrip control

Uniform Resource Name (URN)



Customer branding with the Style Editor works for central CSS only. Imagine that a customer wants to change the light blue color of the standard design to a light yellow all over the portal content area. If you as a developer defined an area with a light blue background directly in your code, the customer has no chance to change this color. Therefore, use central rendering mechanisms only.

3.4.2.3.1.2 HTMLB Controls and Style Editor

Purpose

The look of most visible HTMLB controls can be adapted with the Style Editor. See section *Editability in Style Editor* on the *More Info* page for the respective HTMLB controls.

Controls that use common styles only, such as the standard font color, are not presented in the Style Editor. Examples for these controls are the checkbox, the dropdown list box, and the radio button. The look of these controls can be customized by changing common styles, such as text, links, or cursor definitions.

Browser Platforms

There is a difference with respect to which attributes can be edited or not on different browser platforms. In general, the options for Netscape 4.7 are more restricted than those for other Web browsers.

The following controls cannot be changed for Netscape 4.7 as target platform:

- **Cursor:** Cursor for clickable and non-clickable elements
- **Input Field:** Background color of editable and non-editable input fields
- **Button:** Buttons have the default HTML look (gray and 3D); only the font type and size can be changed via common styles

Uniform Resource Name (URN)

- **Text Edit:** The default HTML element is used; only the font type and size can be changed via common styles

For details with respect to specific controls, see section *Editability in Style Editor* on the *More Info* pages.

Common Styles

There are so-called common styles, which affect more than one control in the content area. We list these styles here, in order to avoid redundant lists for each control.

While for Internet Explorer 5 and above, the common styles affect the controls **cursor**, **input field**, **link** and **text**, for Netscape 4.7 common styles affect only **link** and **text**.

Common styles for controls and different browsers platforms

Control	Style	IE 5 and above	Netscape 4.7
Cursor	Cursor for Clickable Elements	X	
	Cursor for Non-Clickable Elements	X	
Input Field	Background Color for Editable Fields	X	
	Background Color for Non-Editable Fields	X	
Link	Font Color for Unvisited Links	X	X
	Text Decoration for Unvisited Links	X	X
	Font Color for Active Links	X	
	Text Decoration for Active Links	X	
	Font Color for Links on Mouseover (Hover)	X	
	Text Decoration for Links on Mouseover	X	
	Font Color for Visited Links	X	X
	Text Decoration for Visited Links	X	X
Text			
Text Styles	Standard Font Family	X	X
Standard Text	Standard Font Size	X	X
	Standard Font Color	X	X
	Standard Font Style	X	X
	Standard Font Weight	X	X
	Font Size for Small Text	X	X
Non-Standard Text	Font Size for Large Text	X	X
	Font Size for Extra Large Text	X	X
	Font Style for Text Used as a Reference	X	X
	Font Color for Headlines	X	X
	Font Weight for Headlines	X	X
	Font Weight for Emphasized Text	X	X

3.4.2.3.2 Error Handling

Purpose

Beside help, **error handling** is an important aspect of user support. Error handling helps users to overcome problem situations and to continue their work.

Typically, error handling is done by indicating the location where the error occurred and by sending an error message that notes the error, explains the reason for it and - ideally - provides hints on how to remedy the error situation.

For details on message texts see chapter *Formulating Messages* in the *SAP Reference Lists* on the *SAP Design Guild*.

This page covers three areas:

[Error Prevention](#)

[Error Handling for Fields](#)

[Error Handling in Tables](#)

Related Controls

[Flow Layout](#)

[\[Page 33\]Grid Layout \[Page 33\]](#)

3.4.2.3.2.1 Error Prevention

Purpose

Error Prevention Comes First!

Before handling errors, you should first ask how errors can be prevented. Generally, you should design iViews and Web applications so that errors cannot occur. Preventing errors - instead of remedying them - has the following benefits:

- Users cannot come into error situations - many users have problems with recovering from errors.
- The users' work is not interrupted by error messages.
- Users are not confused or puzzled by (often cryptic) error messages.
- There is no need for a screen area that display errors.

If it is not possible to prevent errors, follow the guidelines presented in [Error Handling for Fields \[Page 33\]](#).

How You can Prevent Errors

Often it needs some rethinking and the giving up "old habits" to find design solutions that prevent errors instead of sending an error message **after** an error has occurred.

In the following we provide some ideas and examples that may stimulate your imagination when looking for ways how errors can be prevented.

Prevent Wrong or Invalid Inputs - General

Uniform Resource Name (URN)

- Use precise descriptions and instructions - do not be too short (especially for Web applications)
- Indicate required fields (through a red asterisk *) and an explaining text

Prevent Wrong or Invalid Inputs

- Numeric fields: Prevent users from entering letters by parsing the input string.
- Date and time fields: Provide "intelligent" date and time fields that are preformatted, or provide selection controls instead of input fields (dropdown lists, spin buttons, calendar controls).
- Currency fields: Use preformatted fields.

Prevent Incomplete Inputs

- Indicate required fields (through a red asterisk *) and an explaining text

Prevent Invalid Actions

- Disable buttons that cannot be used in the current context.
- Do not offer functionality that is not needed.

Prevent Disastrous Actions

- If actions can have severe consequences for the user, add explaining texts to the respective buttons and inform the users about the consequences
- Send dialogs if users can loose data

Use Controls in the Correct and Intended Ways

- Do not use screen elements where users expect to use them in any order, if there are dependencies or if a certain sequence of steps has to be followed.



Do not use tabstrips for views that depend on each other and cannot be viewed at random. At best, do not force users to perform steps in a fixed sequence.

- In general, do not use controls in other than the intended ways. "Creative" use of controls clashes with the users' expectations and may lead to severe usage problems.



Do not misuse checkboxes as radiobuttons just because you like the look of the checkboxes better.

Make the Page/View and its Purpose Clear to the User

- Often important information is hidden while unimportant information dominates the page. In other cases users simply have no clue what an application's purpose is. Thus, provide the necessary information and arrange it so that relevant things are recognized first - this way users realize what to do on a screen and how.
- Use precise descriptions and instructions - do not be too short (especially for Web applications)

3.4.2.3.2.2 Error Handling for Fields**Purpose**

Set the field or fields where an error occurred to the error state (see input field) and place an error message as close to the field where the error occurred as possible (if there is more than one field, place the message at the first error field). Place the cursor into the (first) error field.

Avoid Popups!

Popups interrupt the users' work flow and thus annoy them.

Exception: You may use popups for severe errors like aborts that need direct user intervention.

Future Development

After validation of a field, the error message will appear in a line directly below the field. As this change in layout can be performed locally, there will be no major screen flicker.

iViews: In addition, iViews (trays) will have a status bar where a general error message will appear. This status bar may also display warnings and success messages (an icon will indicate the type of the message). The location of the status bar can be either below the title bar or at the bottom of the tray (**open**). The status bar may be hidden by the application.

3.4.2.3.2.3 Error Handling in Tables

Purpose

Errors can appear in table views for different reasons. For example, a user may enter invalid data, or certain items from a set cannot be posted. These cases have to be handled differently.

Input Errors

If a user enters invalid data, highlight the erroneous fields and scroll the table to the first field where an error occurred.

If an error message is needed, place it below the table view or - if possible - in a table row directly below the row where the error(s) occurred.

Future Development

Table views will have a status bar, where the error message will appear. Place the cursor into the error field and scroll the table to make the field visible in case it is hidden from view.

If there is more than one error field, display the message for the first error field, place the cursor into that field and scroll the table to make it visible if necessary.

If the cursor is placed into a subsequent error field, display the message for the respective field. If an error is corrected move the cursor to the subsequent error field if there is one and display the respective error message.

If the focus is outside the table view, display the first error message again.

iViews: In addition the planned status bar of an iView (tray) may display a general error message.

Posting Errors

Posting errors often do not require to cancel the whole posting process. It is only necessary to correct and re-post those items that were erroneous. Therefore, redisplay the table view with the erroneous items only and provide the user with a possibility to correct the items. Place an error message above the table.

Future Development

Place the error message inside the status bar of the table view.

3.4.2.3.3 Accessibility of HTMLB Controls

Use

General Information

This page offers general information for application developers using HTMLB who want to make their Web applications accessible. For details see section *Accessibility* on the *More Info* page for the respective controls.

Most accessibility features are already provided by the central rendering engine of HTMLB. Therefore, application developers only have to add those features that cannot be provided by default.

As an application developer, keep in mind that you cannot affect page elements on the basis of HTML tags or attributes. The only interface to the HTMLB controls is the HTMLB programming interface -- the HTMLB attributes and methods for the respective controls.



Application developers cannot set the *title* attribute of elements in order to extend descriptions, they have to use the *setTooltip* method, instead.

They also cannot set the *summary* attribute of tables, they have to use the *setSummary* method provided by HTMLB.

Descriptions

The central HTMLB rendering engine already provides general descriptions for HTMLB controls, such as the type, the state, and on-screen text. Therefore, application developers only have to complement descriptions in case that users need more specific descriptions or instructions. The descriptions written by the application developers are added to the default descriptions that are provided by the central rendering mechanism.



A button description has to be extended if a button opens a new window.

In general, a description has to be extended if a button introduces an interaction that cannot be recognized by a blind user.

Accessibility Flag

Also note that the resulting description that is sent to the users depends on the state of the accessibility flag:

- If the accessibility flag is **set**, the default description is extended by the description that the application developer provided.
- If the flag is **not set** only the description that the application developer provided is sent to the user.

Keyboard Accessibility

As application developers cannot set HTML attributes directly, they do not have access to the *tabIndex* attribute of elements. Consequently, application developers cannot add elements to the accessibility hierarchy themselves in order to make them keyboard accessible.

Input Elements and Corresponding Labels

Input elements, such as checkboxes, dropdown listboxes, input fields, radiobuttons, and text edit controls need to be connected to a label, so that screen readers recognize the association of the label with the input element. Use the HTMLB label control for this purpose (use method *setLabelFor* for identifying the corresponding control).

The connection between a label and its corresponding input element also simplifies the interaction with the element when using the keyboard or mouse.

References

- SAP Portals Accessibility Guidelines
- API Java Docs

3.4.2.4 Layout

Purpose

This page describes a general strategy for layouting Web pages, applications, and iViews. Layouting a page is not just "throwing" controls on a page. Several aspects have to be considered, such as

- Flow of control - how the user progresses through a page when doing his or her work
- Dependencies - how elements on a page affect each other
- Togetherness - which elements on a page belong to each other, there may be closer and farther relations between elements
- Aesthetics and general Gestalt principles - how information can be effectively communicated visually

There are three steps in layouting - these can be done in the following sequence: Determine the ...

3. Sequence of elements (vertical, horizontal)
4. Nesting of elements
5. Spacing between elements at different hierarchy levels.

The **sequence** takes care of the flow of control, dependencies, and information about which elements belong together - the latter in a more linear fashion. The **nesting** also takes care of dependencies and of togetherness -- but in a hierarchical or top-down fashion. The **spacing** takes care for aesthetics and the proper application of Gestalt principles (mostly togetherness).

Structure of the Layout Section in these Guidelines

This paragraph covers **general** layout aspects, such as the roles of sequence, nesting and spacing. [Layout Hierarchy \[Page 33\]](#) covers the detailed **nesting**, that is, which objects have to be on the same level and which can be nested. The pages on [Flow Layout \[Page 33\]](#), [Grid Layout \[Page 33\]](#), and the pages on **spacing** ([single \[Page 33\]](#) and [grouped \[Page 33\]](#) controls) cover the details of spacing.

[General Page Layout Aspects \[Page 33\]](#)

[Layout Hierarchy \[Page 33\]](#)

[Spacing Between Grouped Controls](#)

[\[Page 33\]](#)[Spacing Between Single Controls \[Page 33\]](#)

3.4.2.4.1 General Page Layout Aspects

The Role of Sequence

The sequence of elements should typically be determined by the **flow of control**, that is, the way how users perform their tasks. Often, however, a task may not be linear or users have to step back because of errors. Here, the page designer has to find a "natural" sequence that fits most users and scenarios.

In addition, conventions, such as the reading direction, play an important role for the arrangement of elements. For Western cultures, the typical arrangement of elements is from left to right and from top to bottom, just like the reading direction. **Dependencies** are also typically communicated this way. "First things first" is also a motto, which expresses that there is a "natural progression" in most things we do. For example, when entering a customer's address we start with the name, which is the main information that determines the remainder of the information - we do not start with the street and house number, even though one might infer the customer's name from that information.

Such a rule may be natural to everybody and most designers follow it without even thinking about it. Problems occur, however, if this rule is **broken**, and the flow or dependencies go into the opposite direction. Such reversals often present severe obstacles for users.

Arranging elements on a page is the first step in page design. This can also be done in a prototypical fashion and tested with users (for example with paper prototypes) without worrying for the details of the page design.

The Role of Nesting

There are two basic ways to visually indicate the relation between elements - closeness and nesting. **Closeness** means that objects, which are located closely together, are perceived as more closely related than objects that are farther apart from each other. Closeness of elements is typically combined with direction to indicate flow of control or dependencies. For example, first you enter a value into a search field (left) and then you click the related Go button next to it (right).

Nesting is used to indicate more complex hierarchical relations and dependencies between objects. Nesting is also a way to hide details from users because users can first deal with the high-level objects and then decide, which one they want to inspect more closely.

Nesting can make pages much more complex than simple sequencing of elements because nesting requires the introduction of borders or other visual separators that may clutter pages visually. Therefore, nesting rules have been established that aim to prevent the creation of overly complex pages (see [Layout Hierarchy \[Page 33\]](#) and the respective controls). Spacing can help to reduce the cluttering effect but often requires more space than is available.

Nesting can also be explored in a prototypical fashion (paper prototypes, HTML prototypes); here, the prototype may already be more detailed than in the initial phase.

The Role of Spacing

Spacing is very importing in communicating which elements belong together; it also affects readability and the ability of users to recognize information on a page.

In general, application developers should not need to bother with the details of spacing, that is, with how many pixels they have to insert between, for example, a button and the border of a group. There are two HTMLB controls, the [grid layout \[Page 33\]](#) and the [flow layout \[Page 33\]](#), which take care for the exact spacing. In addition, containers, such as the tray and the group, also care for the outer spacing.

Note: Currently, the spacing controls do not work as intended. Therefore, developers should consult the pages on the [grid layout \[Page 33\]](#) and on the [flow layout \[Page 33\]](#) for the limitations of these controls.

Only high-level prototypes that intend to offer a realistic preview of a final page need to bother with detailed spacing.

Related Topics

[Forms using Check Boxes \[Page 33\]](#)

[Forms using Radio Buttons \[Page 33\]](#)

[Forms using different list types \[Page 33\]](#) (Drop Down Listbox, Itemlist, Listbox)

3.4.2.4.2 Layout Hierarchy

This page describes the layout hierarchy of Web pages, which defines the options for **nesting** page elements. In short, this page tells designers, which page element can be placed into which container element - including placing containers inside containers.

The layout hierarchy is the basis for establishing textual layout rules for pages and page sections. The main goal of such rules is to prevent overly complex and visually cluttered pages caused by excessive nesting.



These rules do not comprise the **spacing** between and within elements.

[From Containers to the Layout Hierarchy](#)

[\[Page 33\]Layout Hierarchy for iViews and Web Applications](#)

[\[Page 33\]Table Overview of the Layout Hierarchy](#)

[\[Page 33\]General Nesting Rules \[Page 33\]](#)

3.4.2.4.2.1 From Containers to the Layout Hierarchy

Page elements can either be containers or non-containers. Containers can contain other elements, non-containers not. The layout hierarchy described below basically deals with container elements, that is, with elements that can contain other elements including other containers. This is critical because too much nesting can let a page appear visually overloaded.

Application Containers

At the root of the layout hierarchy there is a "root" container that contains the application. In the Web or portal environment, there are two cases to consider:

- The application container is a simple **background**, such as a frame or window. This is the case for the so-called Web applications, including the portal administration applications
- The application container is a **tray** or tile. The container which may have elements and controls on its own; the application that resides inside this container may use the services of the container. From the application's point of view the container is all it knows about -- at least from a design perspective.

Container Controls Inside Applications

Inside an application, container controls define the layout hierarchy of an application. Such containers are:

Uniform Resource Name (URN)

- Areas (Web applications only)
- Tabstrips
- Groups
- Subgroups (group of simple elements with or without heading - not included in a group control)

A Matter of Interpretation - Linear Sequences vs. Sequences of Containers

From a technical point of view, not all of these containers are "real" containers. Areas are subdivisions of an application. That is, areas form a linear sequence within an application. Subgroups are groups of simple page elements that may be introduced by a heading. Typically, they are separated from the remainder of the page by whitespace or separator lines.

From a layout point of view, however, it is easier to view areas and subgroups as "real" containers - for the layout process this does not make any difference. The advantage of regarding these elements as real containers is that a layout can be expressed in a hierarchical or tree-like fashion, which makes it easy to gain an overview of the page or application structure.

Non-Containers

While containers create the "skeleton" of a page, non-containers are the "flesh" of a page. These elements are fields, buttons, selection elements, text units, and tables. As these elements differ in complexity, nesting rules ensure that a page cannot become too complex. For example, table views are similar in complexity to groups and tabstrips. Therefore, they are placed on the same level in the layout hierarchy as these containers. A respective rule that takes this aspect into account would state that table views may not be placed into groups or tabstrips if they are the only control that is inside the container.

Separators

Separators, such as line or whitespace "separate" elements or containers. Therefore, they are difficult to integrate into a hierarchical model of a page layout. They can be viewed as "concluding elements" or "borders" of containers (they are easier to integrate into a "linear" model of the layout).

Note that separators are different. While you would separate containers or elements that are on the same hierarchy level with whitespace, you would use lines because that would introduce unnecessary framing.

Creating the Layout Hierarchy

The layout hierarchy is created by placing containers and simple elements on a page. The rules presented below govern how page elements can be combined, either by sequencing them vertically or horizontally, or by nesting.

Containers may contain containers (nodes), simple elements (leaves), or both. In addition, non-containers may reside on the same hierarchy level as containers. But they are "end nodes" and do not continue the layout hierarchy.



A table view may reside on the same level as a group or a tabstrip

The layout rules presented below specify:

Uniform Resource Name (URN)

- Which containers may contain which other container(s) - including itself
- The specific conditions for the nesting, for example, alone or together with other elements
- How many levels deep the nesting may be
- Which simple elements may be placed into which container - and the specific conditions for this
- Which containers and which simple elements are on the same hierarchy level

3.4.2.4.2.2 Layout Hierarchy for iViews and Web Applications

Depending on the container elements used, different application types can have different layout hierarchies. In the case of the portal environment, there are Web applications and iViews. Both application types use different containers, serve different purposes, and therefore differ in complexity with respect to the layout.

iView

- Tray = iView container
 - Tabstrip - may contain:
 - Group (if it is not the only element)
 - Subgroup
 - Table View (if it is not the only element)
 - Simple Elements
 - Separators
 - Group - may contain:
 - Group (if it is not the only element, different group types only)
 - Subgroup
 - Table View (if it is not the only element)
 - Simple Elements
 - Separators
 - Subgroup - may contain:
 - Simple Elements
 - Separators
 - Table View
 - Simple Elements
 - Separators

Generally, there should not be more than one level of nesting within trays/iViews. Also note that tabstrips may not be nested.

Simple elements are: input fields, selection elements, text, buttons, ...



A similar tree can be created for real iViews based on the elements used.

Web Application

- Application Background = Window/frame background = application container
 - Area - may contain:
 - Tabstrip - may contain:
 - Group (if it is not the only element)
 - Subgroup
 - Table View (if it is not the only element)
 - Simple Elements
 - Separators
 - Group - may contain:
 - Tabstrip (if it is not the only element)
 - Group (if it is not the only element, different group types only)
 - Subgroup
 - Table View (if it is not the only element)
 - Simple Elements
 - Separators
 - Subgroup - may contain:
 - Simple Elements
 - Separators
 - Table View
 - Simple Elements
 - Separators
 - Single elements

Generally, there should not be more than one level of nesting within Web applications. Also note that tabstrips may not be nested.

Simple elements are: input fields, selection elements, text, buttons, ...



A similar tree can be created for real Web applications, based on the elements used.

The critical question for Web applications is, whether single elements and containers other than areas can be placed on the application background. Currently, the application background may not be used for non-container elements (see the *IAC Guidelines* in the *SAP Design Guild*). In R/3 applications, header data may be placed on the application background; there is no such a container concept in R/3 applications as areas.

3.4.2.4.2.3 Table Overview of the Layout Hierarchy

The following table overviews present a more detailed description of the layout hierarchy for iViews and Web applications. Red cells explicitly prohibit certain nestings. Yellow cells

Uniform Resource Name (URN)

indicate cases where elements can be placed into other elements with certain restrictions only (see also the reasons for these rules).

iView

Element below can be placed within Container to the right	Container			
	iView (Tray)	Tabstrip	Group	Subgroup
Tabstrip	yes: together with other elements no: as single element	no	no *	no
Group	yes: together with other elements no: as single element	yes	possible - but use with care! one level at maximum - use different types for the nesting	no
Subgroup	yes	yes	yes	no
Table View	yes: together with other elements no: as single element	yes	no *	no
Text Area, Graphic	yes	yes	yes	no
Separator (White Space, Line)	yes	yes	yes	no
Heading	yes: for subgroup, text area, graphic	yes: subgroup, text area, graphic	yes: subgroup, text area, graphic	yes (as heading for the subgroup)
Field, Selection Element, Icon, Button	yes	yes	yes	yes

Legend

- *) As iViews are simple applications, tabstrips and table views should not be placed into group controls.
- Red cells: Nesting forbidden
- Yellow cells: Nesting allowed under certain conditions only
- The **bold no's** indicate common errors, such as nested tabstrips.

Web Application

Element	Container
---------	-----------

Uniform Resource Name (URN)

below can be placed within Container to the right	Area	Tabstrip	Group	Subgroup
Tabstrip	yes (can be a single element with area header as title)	no	yes: together with other elements no: as single element	no
Group	yes: together with other elements no: as single element	yes	possible - but use with care! one level at maximum - use different types for the nesting	no
Subgroup	yes	yes	yes	no
Table View	yes	yes	yes: together with other elements no: as single element	no
Text Area, Graphic	yes	yes	yes	no
Separator (White Space, Line)	yes	yes	yes	no
Heading	yes: group, text area, graphic	yes: subgroup, text area, graphic	yes: subgroup, text area, graphic	yes
Button	yes	yes	yes	yes
Field, Selection Element, Icon, Button	yes: Header data, group no: other data	yes	yes	yes

Legend

- Red cells: Nesting forbidden
- Yellow cells: Nesting allowed under certain conditions only
- The **bold no's** indicate common errors, such as nested tabstrips.

3.4.2.4.2.4 General Nesting Rules

The following nesting rules are derived from the table overviews above and arranged according to design rationales, such as avoiding too much framing and avoiding redundant headers.

Avoid Redundant Headers

The nesting rules defined for the layout hierarchy strive for **avoiding redundant headings**. Thus, do not place:

- Singular group boxes within areas (Web applications only), or tabstrips

- Singular tables with a table heading within group controls

Avoid Too much Framing (Visual Complexity)

Too many frames and borders make screens visually complex and waste screen space. Thus, do not place:

- Singular tables with a table header within group controls - use a table heading instead
- Singular tabstrips within group controls - Web applications: place them in areas instead; use header texts, or the area header as a title for the tabstrip
- Group controls within group controls - use groups with text headers and separators instead (not forbidden but should be used with care - try to use different types for the nesting)
- Tabstrips within tabstrips - nesting tabstrips is a perfect way of information hiding
- Separator lines between containers or container-like elements.

3.4.2.4.3 Spacing Between Grouped Controls



The values you find in here for spacing and layouting can not be used with the grid layout control currently in usage, and are not meant to be used with it. The grid layout control as the current SAP layouting tool does only support very simple design possibilities. For information how to use the grid layout control see: [Grid Layout - Usage and Types \[Page 33\]](#).

A new form layout control is being developed and will be available latest with the 6.0 version of the portal. The pages about spacing you find under the first point "1. General" have been written to support the development of the form layout tool, and to meet the necessities of future design needs in advance.

This paragraph describes the detailed spacing between grouped controls. For the spacing between single controls, see [Spacing Between Single Controls \[Page 33\]](#).

The following issues are covered:

[Spacing in a Tray \[Page 33\]](#) - the offset between a tray's border and its content

[What's a Correct Spacing Good for \[Page 33\]](#) - the reasoning behind tray offsets and caesuras

[Spacing between Primary and Secondary Groups \[Page 33\]](#) - the spacing between primary and secondary groups, that is, between nested groups

[Spacing between Group Controls with Header and Border \[Page 33\]](#) - this comprises more complex controls and the groups

[Spacing of Elements in Groups \[Page 33\]](#) - the offset within groups, such as the offset between the group border and its content and the group header and its content

[Arranging Groups \[Page 33\]](#) - Alignment of groups and offsets between groups within trays

[Spacing Soft Groups \[Page 33\]](#) - Spacing rules for groupings that do not use a group control as container

The spacing rules in short:

- Offset between tray border/header and content: 5 pixels
- Spacing between primary and secondary groups: 10 pixels
- Spacing between group controls with header and border: 10 pixels

Uniform Resource Name (URN)

- Offset within groups, i.e. between group border and content: 5 pixels
- Spacing within groups: 10 pixels between title and content, 10 pixels between content and buttons
- Spacing between soft groups: 15 pixels horizontally, 30 pixels vertically

In this paragraph you find positive and some negative examples for these cases.

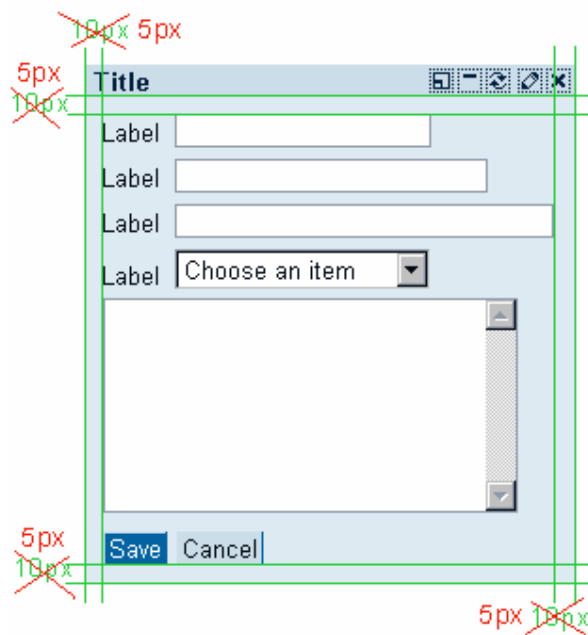
3.4.2.4.3.1 Benefits of Correct Spacing

We use offsets for both groupings and caesuras. In the examples of [Spacing in a tray \[Page 33\]](#) a 5 pixel offset around the tray's content area ensures that a tray's content is realized as being in the tray.

Caesuras separate areas from each other. They stress the individual character of the single area, for instance the group.

3.4.2.4.3.2 Spacing in a Tray

Offset around a tray in EP5.0. The tray offset was only delivered by the grid layout control and thus is only 5 pixels and not 10 as assumed.

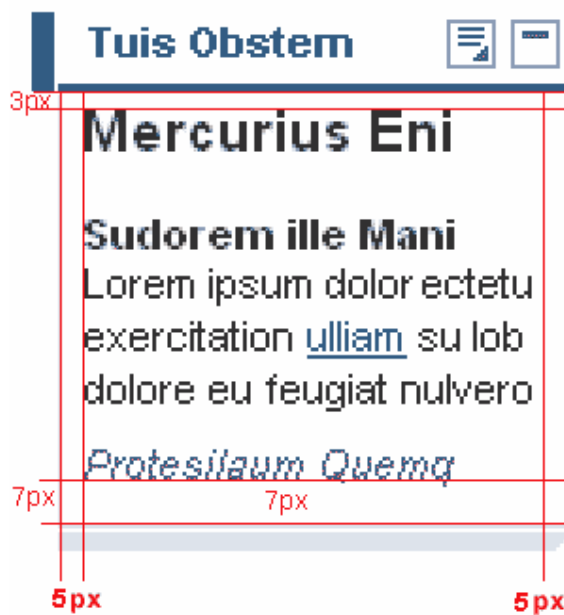


This part of the HTMLB guidelines has been updated. The former specifications for the tray offset are not valid any longer. The old specifications are now canceled and replaced by new ones.

The reason for this is that due to technical reasons in EP5.0 the tray did not deliver the 5 pixel offset it was supposed to, though the grid layout control did so. Thus an offset of only five pixels was possible.

With EP6.0 the offset for the tray content will be delivered by the tray itself. See the further specifications.

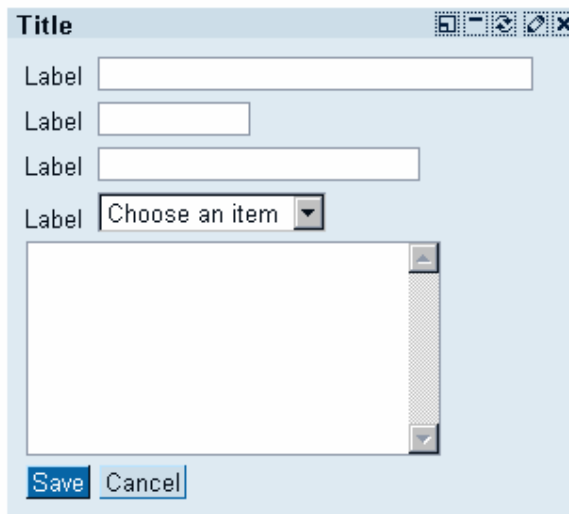
Offset around a tray in EP60



Example of an offset around a tray

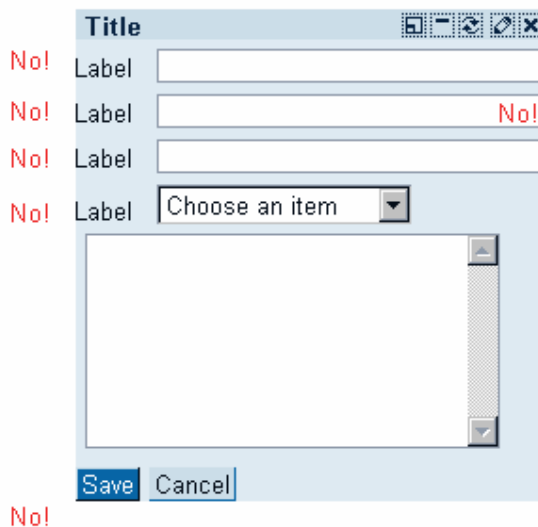
By EP6.0 the whole content offset of a tray will be delivered by the tray itself. With the new design for EP6.0 new design specifications have been made. The new tray offset is specified in the picture on the left.

Uniform Resource Name (URN)



Although you do not always need an offset on the right side, you must always give one to the tray. Whether the offset is needed, depends to the tray's current size which is dependent to the current layout of the portal.

Example of a wrong offset around a tray

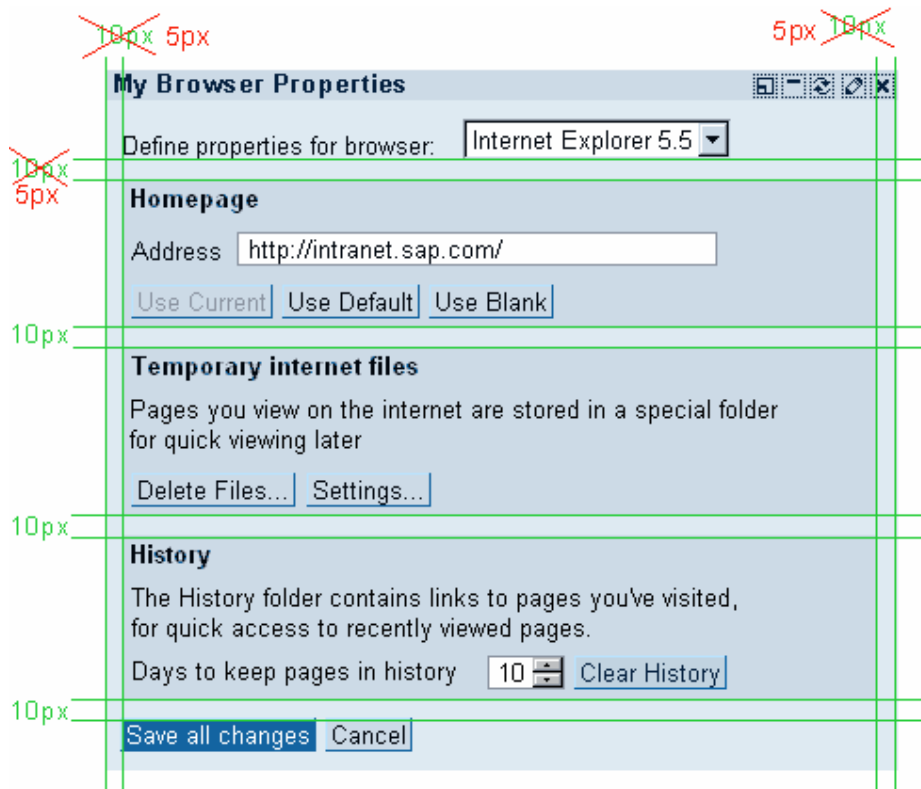


Avoid this. No offset at all gives the impression of elements falling out of the tray.

3.4.2.4.3.3 Spacing between Primary and Secondary Groups

Spacing around secondary groups

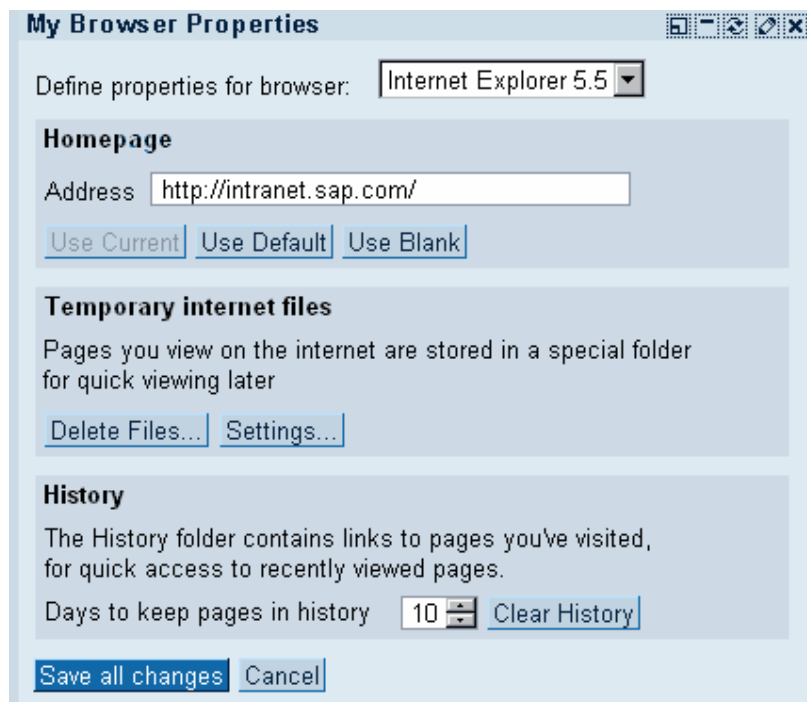
Uniform Resource Name (URN)



The spacing around grouping controls of the type "primary group" and "secondary group", i.e., groups without a border and without a header area, should be 10 pixels.

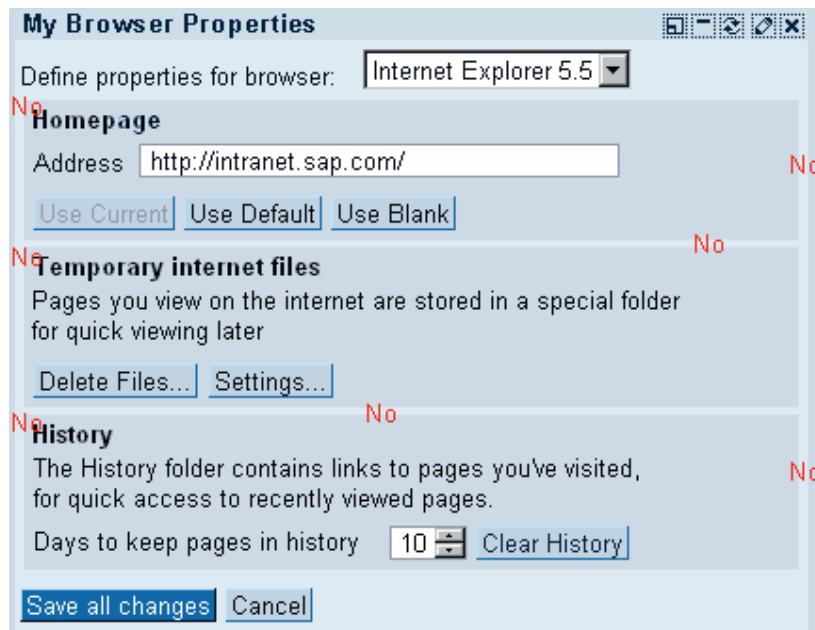
Example of spacing around secondary groups

Uniform Resource Name (URN)



The caesuras clearly stress three areas, realized as three groups.

Example of wrong spacing around secondary groups

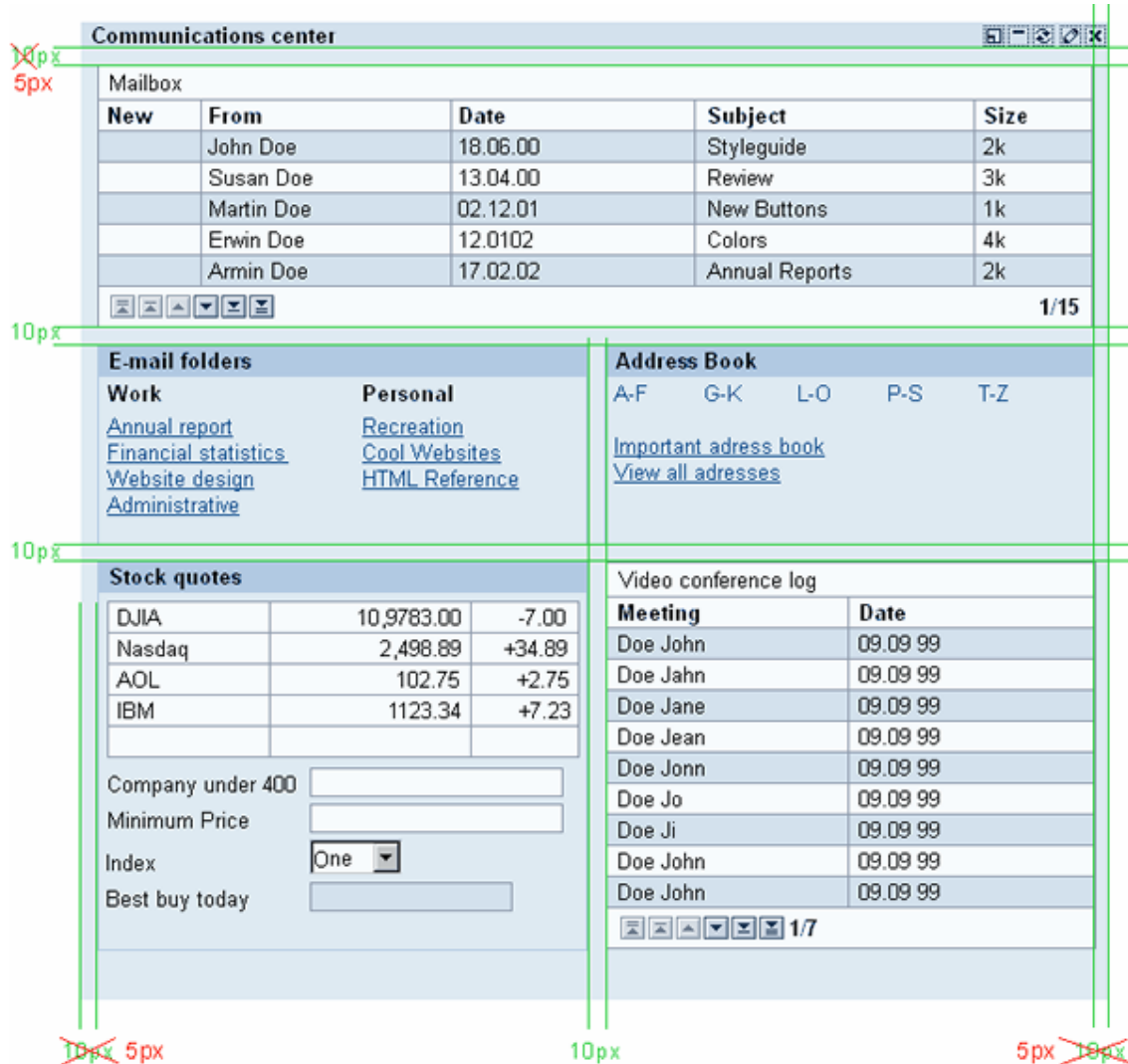


A smaller offset blurs the contrast between secondary group control and the tray's background.

3.4.2.4.3.4 Spacing between Group Controls with Header and Border

Group controls with header and border should also be surrounded by a 10 pixel offset to clearly separate the single groups from each other.

Spacing group controls with header and border



A smaller offset makes the area around the borders noisy and disquiet. When more than two borders come together in a very small space it is very hard to figure out which border belongs to which group. It becomes even harder, when the borders have the same color.

Example of noisy interface

Uniform Resource Name (URN)

Communications center

Mailbox

New	From	Date	Subject	Size
	John Doe	18.06.00	Styleguide	2k
	Susan Doe	13.04.00	Review	3k
	Martin Doe	02.12.01	New Buttons	1k
	Erwin Doe	12.0102	Colors	4k
	Armin Doe	17.02.02	Annual Reports	2k

1/15

E-mail folders

Work	Personal
Annual report	Recreation
Financial statistics	Cool Websites
Website design	HTML Reference
Administrative	

Address Book

A-F G-K L-O P-S T-Z

[Important address book](#)

[View all addresses](#)

Stock quotes

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75
IBM	1123.34	+7.23

Company under 400

Minimum Price

Index

Best buy today

Video conference log

Meeting	Date
Doe John	09.09 99
Doe Jahn	09.09 99
Doe Jane	09.09 99
Doe Jean	09.09 99
Doe Jonn	09.09 99
Doe Jo	09.09 99
Doe Ji	09.09 99
Doe John	09.09 99
Doe John	09.09 99

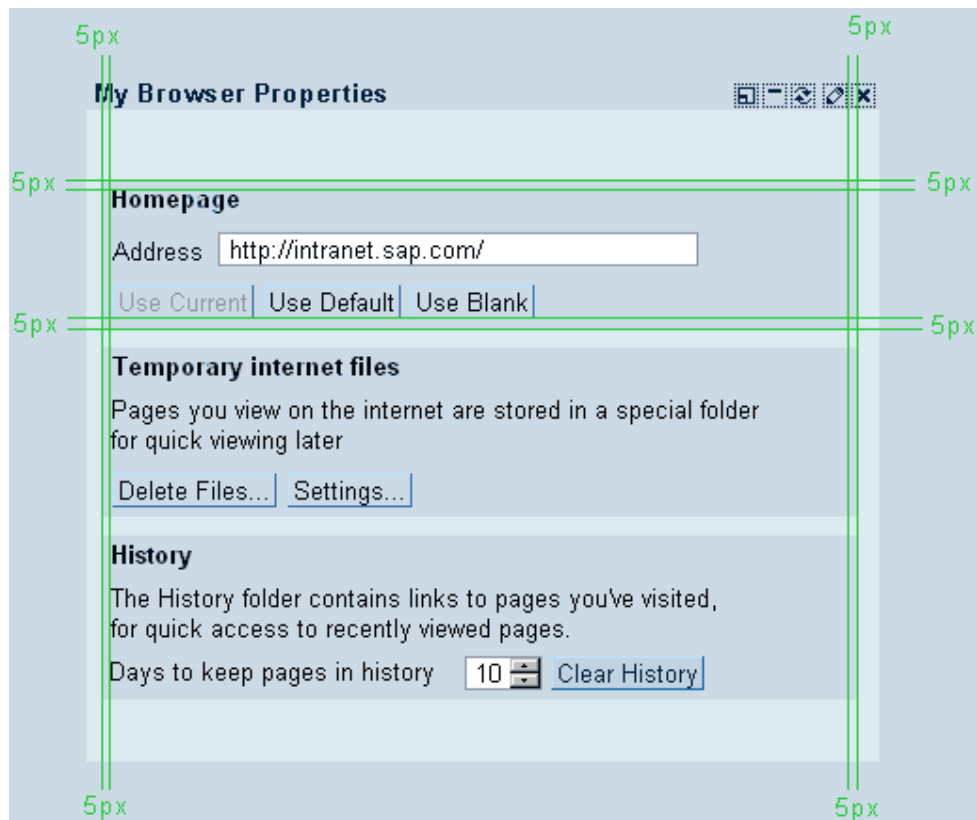
1/7

3.4.2.4.3.5 Spacing of Elements in Groups

A group's content should be surrounded by an offset of five pixels.

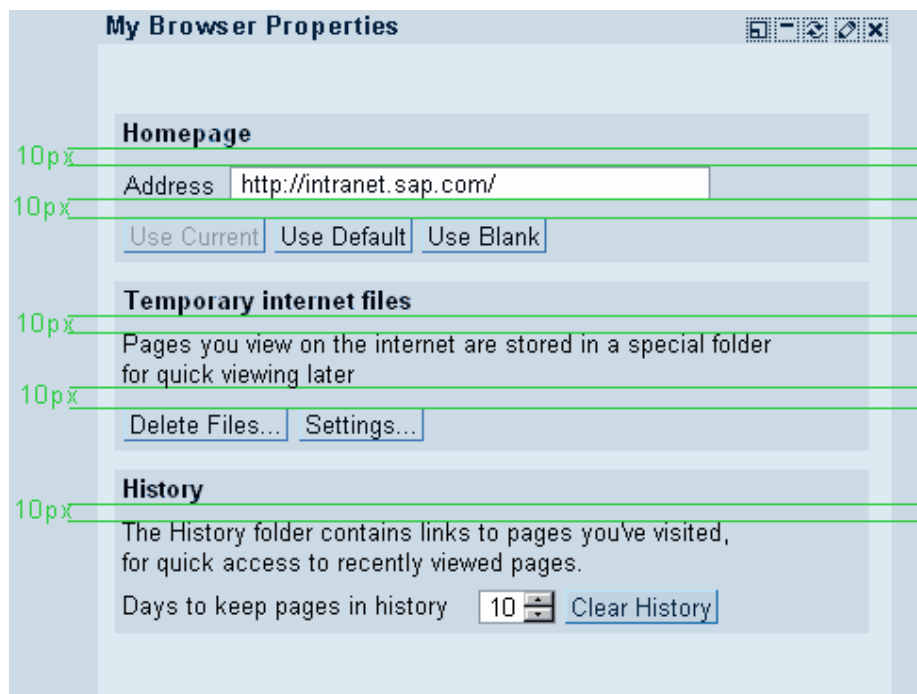
A group's inner offset – borders

Uniform Resource Name (URN)



Leave an offset of 10 pixels beneath titles and above buttons.

A group's inner offset - inner spacing



3.4.2.4.3.6 Arranging Groups

Groups must have a vertical alignment which is achieved by giving them a width of 50%. A horizontal alignment is nice to have but not necessary. However, a scenario like the following must be avoided by any means.

Arranged groups

Communications center

Mailbox

New	From	Date	Subject	Size
	John Doe	18.06.00	Styleguide	2k
	Susan Doe	13.04.00	Review	3k
	Martin Doe	02.12.01	New Buttons	1k
	Erwin Doe	12.0102	Colors	4k
	Armin Doe	17.02.02	Annual Reports	2k

1/15

E-mail folders

Work	Personal
Annual report	Recreation
Financial statistics	Cool Websites
Website design	HTML Reference
Administrative	

Stock quotes

Company	Price	Change
DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75
IBM	1123.34	+7.23

Company under 400

Minimum Price

Index

Best buy today

Video conference log

Meeting	Date
Doe John	09.09 99
Doe Jahn	09.09 99
Doe Jane	09.09 99
Doe Jean	09.09 99
Doe Jonn	09.09 99
Doe Jo	09.09 99
Doe Ji	09.09 99
Doe John	09.09 99
Doe John	09.09 99

1/7

Address Book

A-F G-K L-O P-S T-Z

[Important adress book](#)

[View all adresses](#)

Example of unaligned groups

Communications center

E-mail folders

Work	Personal
Annual report	Recreation
Financial statistics	Cool Websites
Website design	HTML Reference
Administrative	

No

Video conference log

Meeting	Date
Doe John	09.09 99
Doe Jahn	09.09 99
Doe Jane	09.09 99
Doe Jean	09.09 99
Doe Jonn	09.09 99
Doe Jo	09.09 99
Doe Ji	09.09 99
Doe John	09.09 99
Doe John	09.09 99

No

Stock quotes

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75
IBM	1123.34	+7.23

Company under 400

Minimum Price

Index

Best buy today

Address Book

A-F G-K L-O P-S T-Z

No

[Important adress book](#)

[View all addresses](#)

Avoid scenarios of this kind!

3.4.2.4.3.7 Spacing Soft Groups

All those groupings which are not hold together by a second or third control that, on the visual side, is rendered as a box, are called soft groups. Soft groups are formatted text or elements that are both, gathered under a header and separated by caesuras. We separate soft groups from each other by using blank space. The advantage of doing so is, we need less code as we do not use an additional control. Second, we have a quite interface as we do not use group boxes with borders or HTML elements like horizontal rulers. The disadvantage of this method is, we have to waste a lot of space to clearly separate single groups from each other. Use a caesura of 15 pixels to separate soft groups from each other.

Soft groups

Uniform Resource Name (URN)

Hi-tech

☐ **Stocks Retreat in Afternoon as Bonds Slip**
NEW YORK (Reuters) - Wall Street stocks retraced most of their gains in late tracking the bond market lower after a lackluster auction of U.S. Treasury bonds.

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75

☒ **Internet Shares Move Higher Once Again**
NEW YORK (Reuters) - Internet shares moved higher for the second straight day higher interest rates lessened. The American Stock Exchange 48-share Internet

☐ **Internet Shares Mover Higher Once Again**
NEW YORK (Reuters) - The American Stock Exchange 48 - share Internet index points, or 2.54 percent, at 269.39 in early afternoon trading, extending.
To send this article to a friend enter the e-mail address

High Tech Companies Listed Alpabetically

A-C

D-F

G-K

L-O

P-S

T-V

W-Z

Computers & business equipment
Computers
Computer storage devices
Computer peripherals
Office machines consumer
Electronics
Household audio & video equipment
Electronic games and toys
Semiconductors and other
Resistors
Coils, transformers and, inductors
Process and control instruments

Electronic games and toys
Semiconductors and other
Resistors
Coils, transformers and, inductors
Process and control instruments
Laboratory analytical instruments

Industrial Trends

- e-business
- Supply Chain Mgt
- IT
- Knowledge Management
- Globalization
- Channels of Distribution
- Emerging Technologies
- Java
- RosettaNet
- XML

Industrial Processes

- Enterprise Management
- Customer Service
- Product Design
- Supply Chain Planning
- Procurement
- Manufacturing

Industries

- Computer
- Elektronics
- Graphics
- Hardware
- Microcomputers
- Motors
- Networks
- News
- Nutrition
- Plastic
- Strategic Development

Title and Text can be assigned clearly.

Example of soft groups

Uniform Resource Name (URN)

Hi-tech

☐ **Stocks Retreat in Afternoon as Bonds Slip**
NEW YORK (Reuters) - Wall Street stocks retraced most of their gains in late tracking the bond market lower after a lackluster auction of U.S. Treasury bonds.

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75

☒ **Internet Shares Move Higher Once Again**
NEW YORK (Reuters) - Internet shares moved higher for the second straight day higher interest rates lessened. The American Stock Exchange 48-share Internet

☐ **Internet Shares Mover Higher Once Again**
NEW YORK (Reuters) - The American Stock Exchange 48 - share Internet index points, or 2.54 percent, at 269.39 in early afternoon trading, extending.
To send this article to a friend enter the e-mail address

High Tech Companies Listed Alphabetically

[A-C](#)
[D-F](#)
[G-K](#)
[L-O](#)
[P-S](#)
[T-V](#)
[W-Z](#)

[Computers & business equipment](#)
[Computers](#)
[Computer storage devices](#)
[Computer peripherals](#)
[Office machines consumer](#)
[Electronics](#)
[Household audio & video equipment](#)
[Electronic games and toys](#)
[Semiconductors and other](#)
[Resistors](#)
[Coils, transformers and inductors](#)
[Process and control instruments](#)

[Electronic games and toys](#)
[Semiconductors and other](#)
[Resistors](#)
[Coils, transformers and inductors](#)
[Process and control instruments](#)
[Laboratory analytical instruments](#)

Industrial Trends

- [e-business](#)
- [Supply Chain Mgt](#)
- [IT](#)
- [Knowledge Management](#)
- [Globalization](#)
- [Channels of Distribution](#)
- [Emerging Technologies](#)
- [Java](#)
- [RosettaNet](#)
- [XML](#)

Industrial Processes

- [Enterprise Management](#)
- [Customer Service](#)
- [Product Design](#)
- [Supply Chain Planning](#)
- [Procurement](#)
- [Manufacturing](#)

Industries

- [Computer](#)
- [Elektronics](#)
- [Graphics](#)
- [Hardware](#)
- [Microcomputers](#)
- [Motors](#)
- [Networks](#)
- [News](#)
- [Nutrition](#)
- [Plastic](#)
- [Strategic Development](#)

It is possible to use a two column layout like in this example. If doing so, a caesura of 30 pixels should be used.

Caesura between a two column layout

Uniform Resource Name (URN)

Hi-tech

☐ **Stocks Retreat in Afternoon as Bonds Slip**
 NEW YORK (Reuters) - Wall Street stocks retraced most of their gains in late trading the bond market lower after a lackluster auction of U.S. Treasury bonds.

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75

☒ **Internet Shares Move Higher Once Again**
 NEW YORK (Reuters) - Internet shares moved higher for the second straight day higher interest rates lessened. The American Stock Exchange 48-share Internet

☐ **Internet Shares Mover Higher Once Again**
 NEW YORK (Reuters) - The American Stock Exchange 48 - share Internet index points, or 2.54 percent, at 269.39 in early afternoon trading, extending.

To send this article to a friend enter the e-mail address

High Tech Companies Listed Alphabetically

[A-C](#) [D-F](#) [G-K](#) [L-O](#) [P-S](#) [T-V](#) [W-Z](#)

[Computers & business equipment](#) [Electronic games and toys](#)
[Computers](#) [Semiconductors and other](#)
[Computer storage devices](#) [Resistors](#)
[Computer peripherals](#) [Coils, transformers and inductors](#)
[Office machines consumer](#) [Process and control instruments](#)
[Electronics](#) [Laboratory analytical instruments](#)
[Household audio & video equipment](#)
[Electronic games and toys](#)
[Semiconductors and other](#)
[Resistors](#)
[Coils, transformers and inductors](#)
[Process and control instruments](#)

Industrial Trends

- [e-business](#)
- [Supply Chain Mgt](#)
- [IT](#)
- [Knowledge Management](#)
- [Globalization](#)
- [Channels of Distribution](#)
- [Emerging Technologies](#)
- [Java](#)
- [RosettaNet](#)
- [XML](#)

Industrial Processes

- [Enterprise Management](#)
- [Customer Service](#)
- [Product Design](#)
- [Supply Chain Planning](#)
- [Procurement](#)
- [Manufacturing](#)

Industries

- [Computer](#)
- [Elektronics](#)
- [Graphics](#)
- [Hardware](#)
- [Microcomputers](#)
- [Motors](#)
- [Networks](#)
- [News](#)
- [Nutrition](#)
- [Plastic](#)
- [Strategic Development](#)

Whenever we decide to use a layout of this kind, we do not use more than two columns. When using two columns we can decide between dividing the available space in proportions of:

1/2 and 1/2
 or: 2/3 and 1/3
 or: 1/3 and 2/3.

Multiple column layout

Uniform Resource Name (URN)

Hi-tech

☐ **Stocks Retreat in Afternoon as Bonds Slip**
NEW YORK (Reuters) - Wall Street stocks retraced most of their gains in late trading tracking the bond market lower after a lackluster auction of U.S. Treasury bonds.

DJIA	10,9783.00	-7.00
Nasdaq	2,498.89	+34.89
AOL	102.75	+2.75

☒ **Internet Shares Move Higher Once Again**
NEW YORK (Reuters) - Internet shares moved higher for the second straight day higher interest rates lessened. The American Stock Exchange 48-share Internet

☐ **Internet Shares Mover Higher Once Again**
NEW YORK (Reuters) - The American Stock Exchange 48 - share Internet index points, or 2.54 percent, at 269.39 in early afternoon trading, extending.

To send this article to a friend enter the e-mail address
 2/3

High Tech Companies Listed Alphabetically
[A-C](#) [D-F](#) [G-K](#) [L-O](#) [P-S](#) [T-V](#) [W-Z](#)
[Computers & business equipment](#) [Electronic games and toys](#)
[Computers](#) [Semiconductors and other](#)
[Computer storage devices](#) [Resistors](#)
[Computer peripherals](#) [Coils, transformers and inductors](#)
[Office machines consumer](#) [Process and control instruments](#)
[Electronics](#) [Laboratory analytical instruments](#)
[Household audio & video equipment](#)
[Electronic games and toys](#)
[Semiconductors and other](#)
[Resistors](#)
[Coils, transformers and inductors](#)
[Process and control instruments](#)

Industrial Trends

- [e-business](#)
- [Supply Chain Mgt](#)
- [IT](#)
- [Knowledge Management](#)
- [Globalization](#)
- [Channels of Distribution](#)
- [Emerging Technologies](#)
- [Java](#)
- [RosettaNet](#)
- [XML](#)

Industrial Processes

- [Enterprise Management](#)
- [Customer Service](#)
- [Product Design](#)
- [Supply Chain Planning](#)
- [Procurement](#)
- [Manufacturing](#)

Industries

- [Computer](#)
- [Elektronics](#)
- [Graphics](#)
- [Hardware](#)
- [Microcomputers](#)
- [Motors](#)
- [Networks](#)
- [News](#)
- [Nutrition](#)
- [Plastic](#)
- [Strategic Development](#)

3.4.2.4.4 Spacing Between Single Controls

The values you find in here for spacing and layouting can not be used with the grid layout control currently in usage, and are not meant to be used with it. The grid layout control as the current SAP layouting tool does only support very simple design possibilities. For information how to use the grid layout control see: [Grid Layout / Usage and Types \[Page 33\]](#).

A new form layout control is being developed and will be available latest with the 6.0 version of the portal. The pages about spacing you find under the first point "1. General" have been written to support the development of the form layout tool, and to meet the necessities of future design needs in advance.

This page describes the detailed spacing between single controls. For the spacing between grouped controls, see [Spacing Between Grouped Controls \[Page 33\]](#).

The following controls are covered here:

- [Groups of Entry Fields \[Page 33\]](#) - this is the most often needed case for form-line applications
- [Check Box Groups \[Page 33\]](#) and [Radio Button Groups \[Page 33\]](#) - these elements are often used in groups for offering choices or options

Uniform Resource Name (URN)

- [Mixed Form Elements in Vertical Succession \[Page 33\]](#) - this case covers combinations of different input elements, which are arranged in one vertical column; for several columns refer to the spacing for multi-column checkbox/radio button groups

The spacing rules in short:

- Vertical spacing between single elements: 5 pixels for fields and dropdown list boxes, 8 pixels for checkboxes and radio buttons
- Horizontal spacing between multiple columns: 15 pixels
- Horizontal spacing between label and input element, width of label column: Width of widest label plus an offset of 8 to 22 pixels
- Spacing between selection element and label: 8 pixels for checkboxes and radio buttons

In this paragraph you find positive and negative examples for all these cases.

3.4.2.4.1 Groups of Entry Fields

Offset between fields

The screenshot shows a form titled 'Create an Account' with a toolbar at the top right. The form contains the following labels and input fields: First Name, Last Name, Preferred Greeting Name, E-mail Address, Re-enter E-mail Address, Password, Re-enter Your Password, and Birth Date. A blue 'Continue' button is at the bottom left. Green horizontal lines with '5px' labels indicate the consistent vertical offset between the bottom of one input field and the top of the next.

Leave an offset of five pixels between entry fields.

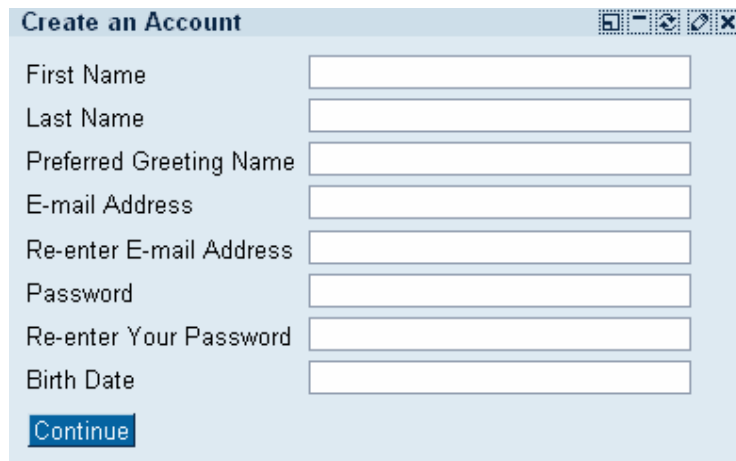
Fields have ragged edges

This screenshot shows the same 'Create an Account' form, but the input fields are left-aligned with varying widths, resulting in ragged right edges. The labels and the 'Continue' button are in the same positions as in the previous screenshot.

At SAP it is common style to have fields that are left aligned with ragged edges on the right side.

Example of justified fields

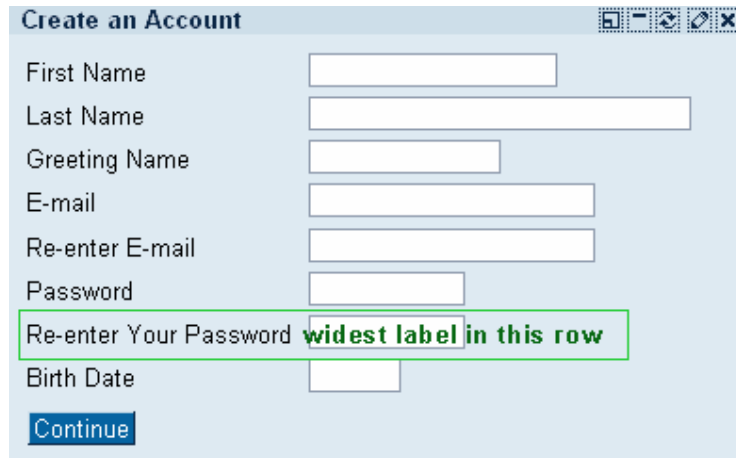
Uniform Resource Name (URN)



The screenshot shows a web form titled "Create an Account" with a light blue background. It contains eight input fields, each preceded by a label: "First Name", "Last Name", "Preferred Greeting Name", "E-mail Address", "Re-enter E-mail Address", "Password", "Re-enter Your Password", and "Birth Date". The labels are left-aligned, and the input fields are right-aligned, creating a justified appearance. A blue "Continue" button is located at the bottom left of the form.

Justified fields are not necessarily wrong. However, it is hard to figure out why one needs a birth date field with more than eight characters.

Offset between label and fields



This screenshot shows the same "Create an Account" form, but with an offset between the labels and the input fields. The labels are left-aligned, and the input fields are right-aligned, but there is a consistent gap between them. A green box highlights the "Re-enter Your Password" label and its corresponding input field, with the text "widest label in this row" written in green next to the label. A blue "Continue" button is at the bottom left.

As one can never predict the length of a field label on the one side, and how many fields will be necessary in one scenario in succession on the other, it is hardly possible to give a standard offset between label and entry field. As a rule of thumb one can say: In one row of entry fields that follow each other in succession consider the offset between the widest label and its entry field. If possible, try to avoid an offset smaller than 8 pixels, which is one character, and wider than 22 pixels, which is three characters. In the scenario on the left the offset between widest label and its corresponding entry field is 8 pixels.

Offset between label and fields

Uniform Resource Name (URN)

The screenshot shows a 'Create an Account' form with the following labels and input fields:

- First Name: [input field]
- Last Name: [input field]
- Greeting Name: [input field]
- E-mail: [input field]
- Re-enter E-mail: [input field]
- Password: [input field]
- Re-enter Your Password: [input field]
- Birth Date: [input field]

A green double-headed arrow indicates the offset between the 'E-mail' label and its input field. A green box highlights the 'Re-enter Your Password' label and input field with the text: **Here the offset is 22 pixels**. A 'Continue' button is at the bottom.

By restricting the space next to the widest label to a maximum size we ensure that the offset between the smallest label and its corresponding entry field is not too large and the user can still adjust label and entry field to each other.

Example of too large offset between label and fields

The screenshot shows the same 'Create an Account' form as above, but with a significantly larger offset between labels and input fields. The text **Avoid this** is written in red at the bottom right of the form area. A 'Continue' button is at the bottom.

Here you can still adjust the largest label and its corresponding field but it becomes almost hard work adjusting "E-mail" to its input field.

Example of too small offset between label and fields

The screenshot shows the same 'Create an Account' form as above, but with a significantly smaller offset between labels and input fields. The text **Avoid this** is written in red at the bottom right of the form area. A 'Continue' button is at the bottom.

Though all offsets seem to look correct the missing offset between "Reenter Your Password" and its entry field makes the whole interface look ugly.

3.4.2.4.4.2 Check Box Groups

Leave an offset of eight pixels between check boxes and their corresponding label.

Offset between check boxes and their labels

Sign up for your site - step 3

Contact me occasionally about special offers, promotions and features.

Interests (optional):

<input checked="" type="checkbox"/> Entertainment	<input checked="" type="checkbox"/> Business	<input checked="" type="checkbox"/> Shopping
<input checked="" type="checkbox"/> Home & Family	<input checked="" type="checkbox"/> Computers & Technology	<input checked="" type="checkbox"/> Sport & Outdoors
<input checked="" type="checkbox"/> Health	<input checked="" type="checkbox"/> Personal Finance	<input checked="" type="checkbox"/> Travel
<input checked="" type="checkbox"/> Music	<input checked="" type="checkbox"/> Small Business	<input checked="" type="checkbox"/> Sweepstakes & Free Stuff

8px 8px 8px

Leave an offset of eight pixels between rows of check boxes.

Offset between rows of check boxes

Sign up for your site - step 3

Contact me occasionally about special offers, promotions and features.

Interests (optional):

<input checked="" type="checkbox"/> Entertainment	<input checked="" type="checkbox"/> Business	<input checked="" type="checkbox"/> Shopping
<input checked="" type="checkbox"/> Home & Family	<input checked="" type="checkbox"/> Computers & Technology	<input checked="" type="checkbox"/> Sport & Outdoors
<input checked="" type="checkbox"/> Health	<input checked="" type="checkbox"/> Personal Finance	<input checked="" type="checkbox"/> Travel
<input checked="" type="checkbox"/> Music	<input checked="" type="checkbox"/> Small Business	<input checked="" type="checkbox"/> Sweepstakes & Free Stuff

8px 8px 8px

Leave an offset of 15 pixels between columns of check boxes.

Offset between groups of check boxes

Sign up for your site - step 3

Contact me occasionally about special offers, promotions and features.

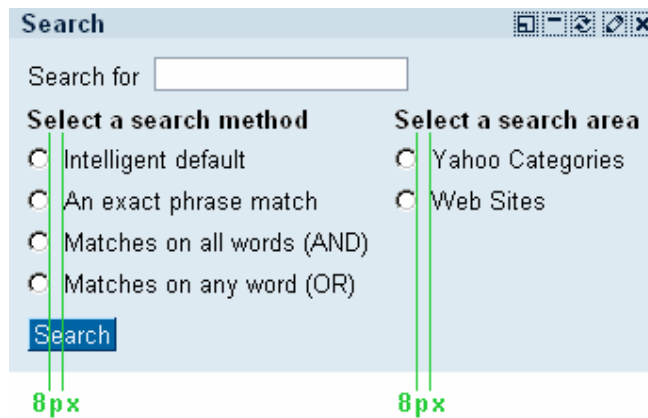
Interests (optional):

<input checked="" type="checkbox"/> Entertainment	<input checked="" type="checkbox"/> Business	<input checked="" type="checkbox"/> Shopping
<input checked="" type="checkbox"/> Home & Family	<input checked="" type="checkbox"/> Computers & Technology	<input checked="" type="checkbox"/> Sport & Outdoors
<input checked="" type="checkbox"/> Health	<input checked="" type="checkbox"/> Personal Finance	<input checked="" type="checkbox"/> Travel
<input checked="" type="checkbox"/> Music	<input checked="" type="checkbox"/> Small Business	<input checked="" type="checkbox"/> Sweepstakes & Free Stuff

15px 15px

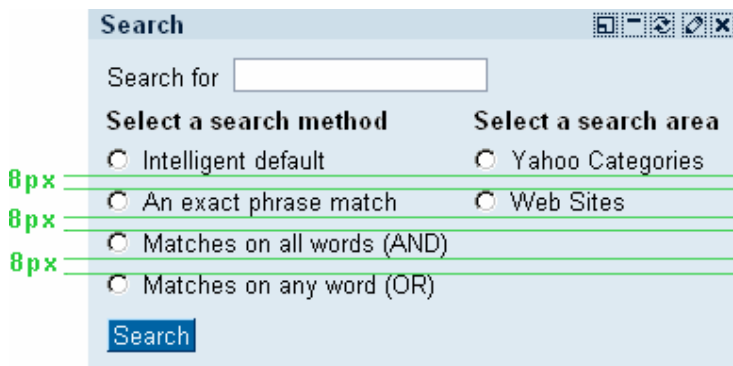
3.4.2.4.4.3 Radio Button Groups

Offset between radio buttons and their labels



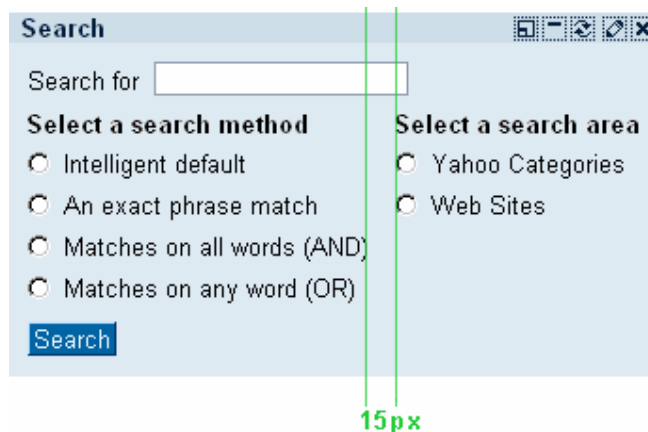
Leave an offset of eight pixels between radio buttons and their corresponding label.

Offset between rows of radio buttons



Leave an offset of eight pixels between rows of radio buttons.

Offset between radio button groups



Leave an offset of 15 pixels between columns of radio buttons.

3.4.2.4.4.4 Mixed Form Elements in Vertical Succession

Offset around mixed form elements in vertical succession

Uniform Resource Name (URN)

The 'System Info' dialog box contains the following elements:

- Type in a system:
- And launch: Transaction
- With name: SE80
- In client: 001
- In gui: ☐ First gui, ☐ Second gui, ☐ Third gui
- With language: English
- Show landscape as: ☒ Flash animation, ☒ Generated list, ☒ Static graph
- Buttons: Continue, Cancel

Green arrows on the right indicate 5px vertical offsets between the following elements:

- Between 'Type in a system' and 'And launch'
- Between 'And launch' and 'With name'
- Between 'With name' and 'In client'

Between field like form elements in a vertical succession is always an offset of 5 pixels.

Offset around mixed form elements in vertical succession

This screenshot is identical to the one above, showing the 'System Info' dialog box with its various form elements and 5px vertical offsets between field-like elements.

Above and beneath button like form elements is always an offset of 8 pixels regardless of the following or previous element.

Offset between horizontal groups of mixed form elements in vertical succession

Uniform Resource Name (URN)

System Info

Type in a system

And launch

With name

In client

In gui ☐ First gui
☐ Second gui
☐ Third gui

With language

Show landscape as ☒ Flash animation
☒ Generated list
☒ Static graph

15px

Leave an offset of 15 pixels between columns of form elements.

3.4.2.5 Layout Controls

Purpose

To layout a page is not just "throwing" controls on a page. Several aspects have to be considered, such as:

- Flow of control
How the user progresses through a page when doing his or her work.
- Dependencies
How elements on a page affect each other.
- Togetherness
Which elements on a page belong to each other, there may be closer and farther relations between elements.
- Aesthetics and general design principles
How information can be effectively communicated visually.

The layout process has three steps - these can be done in the following sequence:

- Sequence of elements (vertical, horizontal).
- Nesting of elements.
- Spacing between elements at different hierarchy levels.

The sequence takes care of the flow of control, dependencies, and information about which elements belong together - the latter in a more linear fashion. The nesting also takes care of dependencies and of togetherness - but in a hierarchical or top-down fashion. The spacing takes care for aesthetics and the proper application of design principles (mostly togetherness).

Layout Controls

[Content \[Page 33\]](#)

[Document](#)

[\[Page 33\]Page](#)

[\[Page 33\]Form](#)

[\[Page 33\]Flow Layout](#)

[\[Page 33\]Form Layout](#)

[\[Page 33\]Grid Layout \[Page 33\]](#)

3.4.2.5.1 Content

Definition

The Control API creates a scripting variable which provides the rendering context for the following tags. It is a plain HTML tag.

- **id**
Identification name of the content.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="myContent" <i>Classlib</i> Set with control document [Page 33]

Example

```
<hbj:content id="myContent">
    ...
</hbj:content>
```

3.4.2.5.2 Document

Definition

Renders the root tag of the document (for example, <html> or <wml>) depending on the markup language used. It is a plain HTML tag with no attributes. The document control has two additional controls:

[documentBody \[Page 33\]](#): Renders the <body> section of the document

[documentHead \[Page 33\]](#): Renders the <head> section of the document

Attributes	M	Values	Usage
------------	---	--------	-------

Uniform Resource Name (URN)

id	*	String (cs)	<i>Taglib</i> Defined with content [Page 33] tag. <i>Classlib</i> setDocumentId ("myContent")
title	*	String	<i>Taglib</i> Defined with documentHead [Page 33] tag. <i>Classlib</i> setTitle ("SAP")

Example

```
<hbj:content id="myContent">
  <hbj:document>
    ...
  </hbj:document>
</hbj:content>
```

3.4.2.5.2.1 DocumentBody**Definition**

Renders `<body>` section of the document and attaches the appropriate style class. It is a plain HTML tag with no attributes.

Example

```
<hbj:content id="myContent">
  <hbj:document>
    <hbj:documentBody>
      ...
    </hbj:documentBody>
  </hbj:document>
</hbj:content>
```

3.4.2.5.2.2 DocumentHead**Definition**

Renders `<head>` section of the document and includes the necessary style sheets and scripts. It is a plain HTML tag. In the documentHead a nested **META** element (standard HTML) can be used. With **META** element information about the document (name, content, scheme, http-equiv) can be specified.

Uniform Resource Name (URN)

- **title**
Set the title that is usually displayed in the title bar of the web client.

Attributes	M	Values	Usage
title	*	String	<i>Taglib</i> title="SAP" <i>Classlib</i> See 'document' [External] control

Example

```
<hbj:content id="myContent">
  <hbj:document>
    <hbj:documentHead title="sap">
      <meta name="description" content="Introduction page">
      <meta name="author" content="sap">
      <meta name="date" content="Jan. 2002">
      ...
    </hbj:documentHead>
    <hbj:documentBody>
      ...
    </hbj:documentBody>
  </hbj:document>
</hbj:content>
```

3.4.2.5.3 Page

Definition

Represents a complete HTML page consisting of tags <html>, <head> and <body> and includes the necessary style sheets and scripts. It is a plain HTML tag.



If JavaScripts are used (for 'onClientClick' events) the page tag is necessary for the renderer to place the JavaScripts at the end of the page.

- **title**
Set the title that is usually displayed in the title bar of the web client.

Attributes	M	Values	Usage
title		String	<i>Taglib</i> title="sap" <i>Classlib</i> See document [Page 33]

Example

```
<hbj:content id="myContent">
  <hbj:page title="sap">
    ...
  </hbj:page>
</hbj:content>
```

3.4.2.5.4 Form

Definition

Is the outer shell of the document and encapsulates normal content, markup, controls and labels of those controls. Forms are essential for the event handling.

- **action**
Defines the form processing agent. For example, the value might be a HTTP URI to submit the form to a program or a mailto URI to email the form.
- **defaultButton**
Defines the default button for this document. This button will fire an event if the user presses the RETURN / ENTER key in the web agent. Usually the button should have the 'design' "EMPHASIZED" to graphically show that this button is the default button. If you write an application for different web clients you should be aware of the fact that every web client has its own way to handle keyboard input. To achieve the right results on all web clients you should use the default button always together with an inputField and the inputField must have the focus (usually the inputField gets the focus because the user clicked on this field to do some input). In this case the onClick event of the default button will be fired when the user presses RETURN / ENTER in the inputField.
- **encodingType**
Defines the content type (MIME type) used to submit the form to the web server. Examples of content types can be found at internet address www.w3.org/TR/1998/REC-html40-19980424/interact/forms.html#h-17.3



If you use a fileUpload control in the JSP you must set the encodingType attribute to "multipart/form-data".

Example:

```
<hbj:form encodingType="multipart/form-data">
```

- **focusedControl**
Defines the control which has the focus when the page is loaded. The attribute 'focusedControl' for buttons works like the attribute 'defaultButton'. As extension to the 'defaultButton' attribute, the 'focusedControl' attribute can set the focus to dropdownListBoxes, inputFields and textEdit controls.
- **id**
Identification name of the form.
- **language**
Defines the language code for this document. The attribute defines the primary language and can define a series of sub languages. The language code is according to ISO 639. The language code and the description can be found at internet address

Uniform Resource Name (URN)

www.w3.org/TR/1998/REC-HTML40-19980424/references.html#ref-RFC1766

- **messageBar**
Sets a [messageBar \[Page 33\]](#) for the form.
- **messageBarAtFormEnd**
A boolean value that sets the messageBar at the bottom of the form.
- **messageBarRequired**
A boolean value that defines if a messageBar is required for this form.
- **method**
Defines the HTTP method that will be used to submit the form data set. The form data set is a sequence of control name/current value pairs constructed from successful controls. A successful control is "valid" for submission. Every control has its control name paired with its current value as part of the submitted form data set. A successful control must be defined within a form and must have a control name.



The control name is generated by the HTML-Business for Java renderer. So you have no way to address the control via for example, JavaScript.

- GET
The form data set is appended to the URI specified by the action attribute (with a question-mark (?) as separator) and this new URI is sent to the processing web agent.
- POST
The form data set is included in the body of the form and sent to the processing web agent.

The default method is POST and should not be altered (Limits of GET requests can cause problems).
- **scrollingToLastPosition**
A boolean value that controls the position in a form. By default the position in a form is always reset to "top of form" when the form is submitted (for example, in case of an event). If the 'scrollingToLastPosition' attribute is set to true the last position in the form is saved and restored on a submit.
- **target**
Specifies the name of the frame where the document is to be opened. The following values refer to w3c HTML-standard.
 - _blank
The web client should load the designated document in a new, unnamed window.
 - _self
The web client should load the document in the same frame as the element that refers to the target.
 - _parent
The web client should load the document into the immediate FRAMESET parent of the current frame. This value is equivalent to _self if the current frame has no parent.
 - _top
The web client should load the document into the full, original window (thus canceling all other frames). This value is equivalent to _self if the current frame has no parent.

Uniform Resource Name (URN)

- **userDefinedMessageBar**

A boolean value that defines if a user defined messageBar is used for this form.

Attributes	M	Values	Usage
action		String (cs)	<i>Taglib</i> action="/servlet/test.JspTest" <i>Classlib</i> setAction("/servlet/test.JspTest")
defaultButton	*	String (cs)	<i>Classlib</i> setDefaultButton(Button OKButton)
encodingType		String	<i>Taglib</i> encodingType="multipart/mime" <i>Classlib</i> setEncodingType("multipart/mime")
focusedControl		Component	<i>Classlib</i> setFocusedControl(Button OKButton) setFocusedControl(DropdownListBox dlb) setFocusedControl(InputField inp_field) setFocusedControl(TextEdit text_edit)
id	*	String (cs)	<i>Taglib</i> id="SAPForm"
language		String	<i>Taglib</i> language="de" <i>Classlib</i> setLanguage("de")
messageBar		Component	<i>Classlib</i> setMessageBar (MessageBar [Page 33] bar)
messageBarAtFormEnd		FALSE (d) TRUE	<i>Classlib</i> setMessageBarAtFormEnd(true)
messageBarRequired		FALSE (d) TRUE	<i>Classlib</i> setMessageBarRequired(true)
method		POST (d) GET	<i>Taglib</i> method="POST" <i>Classlib</i> setMethod("POST")
scrollingToLastPosition	*	FALSE (d) TRUE	<i>Classlib</i> setScrollingToLastPosition(true)
target		_blank _self (d) _parent _top	<i>Taglib</i> target="_blank" <i>Classlib</i> setTarget("_blank")
userDefinedMessageBar		FALSE (d) TRUE	<i>Classlib</i> setUserDefinedMessageBar(true)

Uniform Resource Name (URN)

Example

This example also shows the definition of a default button. We define the `OKbutton` as default.

The assignment is made by scripting (`<% %>`) and has to be done where the button is defined.

```
<hbj:form
id="myFormId"
method="post"
target="_blank"
encodingType="multipart/form-data"
action="/htmlb/servlet/com.sap.htmlb.test.MyTestJsp1Test">
This form submits to a new web client window
<br>
because of 'target=_blank'.
<br>
<br>
<hbj:inputField
  id="myInputField1"
  type="String"
  invalid="false"
  width="310"
  value="After editing press <Enter> to submit"
  visible="true"
  disabled="false"
  required="true"
  maxLength="30"
  size="50">
</hbj:inputField>
<br>
<br>
<hbj:button
  id="oKbutton"
  text="OK"
  onClick="onOKClick"
  design="EMPHASIZED"
  width="100">
<%
  myFormId.setDefaultButton(oKbutton);
%>

</hbj:button>
<hbj:button
  id="Infobutton"
  text="Info"
  onClick="onInfoClick"
  width="100"
/>
<hbj:button
  id="CancelButton"
  text="Cancel"
  onClick="onCanClick"
  width="100"
/>
</hbj:form>
```

Result

This form submits to a new web client window because of 'target=_blank'.

After editing press <Enter> to submit

OK

Info

Cancel

3.4.2.5.5 Flow Layout

Control API

The `FlowLayout` is a simple container that renders its contents without additions. It allows to align controls that do not need to be aligned with other elements in an interface. Controls that are added to the `FlowLayout` are able to wrap if the available space for displaying all controls in one line does not suffice.

`FlowLayout` has no tag. It has one method **getUI** which returns an identification string for the renderer that is unique for all supported components. The following attributes are inherited from **container**.

Attribute	M	Description	Case sens	Classlib
<code>addComponent</code>		Adds a component to the container. The component is added to the end of the already added text/components.	yes	<code>addComponent</code> ((component) component)
<code>addRawText</code>		Adds text - without encoding - to the container for example, if HTML commands have to be added. The text is added to the end of the already added text/components.	-	<code>addRawText</code> (java.lang.String text)
<code>addText</code>		Adds text encoded to the container. The text is added to the end of the already added text/components.	-	<code>addText</code> (java.lang.String text)
<code>removeComponent</code>		Removes a component from the container	yes	<code>removeComponent</code> ((component) component)

FlowLayout method:

- **wrapping**
Sets the line wrapping for the flow control.

Attributes	M	Values	Usage
<code>wrapping</code>		FALSE (d) TRUE	<i>Taglib</i> <code>wrapping="TRUE"</code>

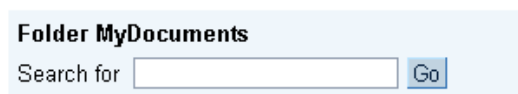
			<i>Classlib</i> <code>setWrapping(true)</code>
--	--	--	---

3.4.2.5.5.1 Usage & Type

Definition

The flow layout is an invisible control used to combine other controls one after another. It can be inserted into every container control.

Example of the usage of flow layout



The controls within a flow layout



The controls will wrap to fit the size of its container (in this case a group).



Use

Use the flow layout if you do not need to align controls with other elements in your interface. Controls that are added to the flow layout are able to wrap if the available space for displaying all controls in one line does not suffice.

To separate controls within the flow layout you should currently use a [Text View \[Page 33\]](#) control containing a simple space character.

When use the Flow Layout - When Use the Form Layout

- Use the flow layout if you do not need to align controls with other elements in your interface; this will enhance performance because the flow layout does not have an overhead of table structures in the rendering
- Use the form layout to align controls with respect to other controls in the user interface

Related Controls

[Form Layout \[Page 33\]](#)

[Grid Layout \[Page 33\]](#)

[Text View \[Page 33\]](#)

Further Information

[Browser Support & Accessibility \(508\) \[Page 33\]](#)

[Control API \[Page 33\]](#)

3.4.2.5.2 Browser Support & 508

The flow layout control structures elements in every browser.

Editability in Style Editor

Currently, the flow layout is not changeable by the Style Editor.

Accessibility - 508 Support

The flow layout has no special accessibility enhancements. It can contain several controls that are by themselves in the accessible hierarchy and might have further descriptions for blind users.

3.4.2.5.6 Form Layout

Control API

A grid is a two dimensional arrangement of data in rows and columns. The control is similar to gridLayout but has more features in adjusting the cells of the grid. If no formLayoutCells are defined no formLayout is displayed.

Limitation:

Large formLayouts (large amount of rows and columns) can cause problems like:

- Compiler errors that are caused by a 64kB method length limit.
- Slow processing of page because of huge HTML-Code generated by the JSP-Compiler
- **debugMode**
A Boolean value. If set to "TRUE" the formLayoutCell is rendered with a frame. The frame size is defined by formLayoutCell 'width' and the padding. If a formLayoutCell is not defined or empty no frame is displayed.
If set to "FALSE" no frame is rendered.

By default the borders of the grid are invisible. To see the borders for layout and debug reasons set the debug attribute..



Setting the debugMode attribute will add pixels to visualize borders. Therefore the sizes of the grid layout will change if you reset the attribute. The debugMode

Uniform Resource Name (URN)

attribute, as indicated by the name, should only be used for debugging and not for "styling".

- **id**
Identification name of the formLayout. You have to specify an id if you want to access the control.
- **marginBottom**
Specifies the distance from the bottom of the control to the next control.
- **marginLeft**
Specifies the distance from the "actual" position to the left side of the control.
- **marginRight**
Specifies distance from the right side of the control to the next control.
- **marginTop**
Specifies distance from the "actual" position to the top of the control.
- **width**
Defines the width of the formLayout. If the 'width' in formLayoutCell is specified in percent, the percentage will be calculated from the width of the formLayout. If the formLayoutCell definition exceeds the 'width' of the formLayout the formLayoutCell content will be wrapped.
-

Attribute	M	Values	Usage
debugMode		FALSE (d) TRUE	<i>Taglib</i> debugMode="TRUE" <i>Classlib</i> setDebugMode(true)
id	*	String	<i>Taglib</i> id="ZIPCode_form" <i>Classlib</i> setId("ZIPCode_form")
marginBottom		Unit (0)	<i>Taglib</i> marginBottom="5" <i>Classlib</i> setMarginBottom("5")
marginLeft		Unit (0)	<i>Taglib</i> marginLeft="5" <i>Classlib</i> setMarginLeft("5")
marginRight		Unit (0)	<i>Taglib</i> marginRight="5" <i>Classlib</i> setMarginRight("5")
marginTop		Unit (0)	<i>Taglib</i> marginTop="5" <i>Classlib</i> setMarginTop("5")
width		Unit (100%)	<i>Taglib</i>

Uniform Resource Name (URN)

			<pre>width="500" Classlib setWidth("500")</pre>
--	--	--	--

formLayoutRow

Defines the rows in the formLayout. The cells (formLayoutCell) have to be nested in form layout rows.

- **id**
Identification name of the formLayoutRow. You have to specify an id if you want to access the control.
- **paddingBottom**
Specifies the bottom padding of each row in the form layout. The value of the paddingBottom attribute represents the distance from the bottom border of the cell to the bottom of the content of each row specified in pixels.
- **paddingTop**
Specifies the top padding of each row in the form layout. The value of the paddingTop attribute represents the distance from the top border of the cell to the top of the content of each row specified in pixels.

Attribute	M	Values	Usage
Id		String (cs)	<i>Taglib</i> <pre>id="ZIPCode_row01"</pre> <i>Classlib</i> <pre>setId("ZIPCode_row01")</pre>
padding		Numeric (0)	<i>Classlib</i> <pre>setPadding(String top, String bottom)</pre> for example: <pre>setPadding("5", "3")</pre>
paddingBottom		Numeric (0)	<i>Taglib</i> <pre>paddingBottom="3"</pre> <i>Classlib</i> <pre>setPaddingBottom("3")</pre>
paddingTop		Numeric (0)	<i>Taglib</i> <pre>paddingTop="5"</pre> <i>Classlib</i> <pre>setPaddingTop("5")</pre>

formLayoutCell

Defines the cells in the formLayoutRow.

- **align**
Defines the horizontal alignment of the cell content.
 - LEFT
Left justifies the content of the cell.

Uniform Resource Name (URN)

- **RIGHT**
Right justifies the content of the cell.
 - **CENTER**
Centers the content of the cell.
 - **CHAR**
Aligns text around a specific character. Not supported by all web clients.
 - **JUSTIFY**
Sets text in the cell left and right aligned. Not supported by all web clients.
- **colspan**
Defines the horizontal expansion the cell in columns.
- **content**
Specifies the content for the cell.
- **id**
Identification name of the formLayoutCell.
- **paddingBottom**
Specifies the bottom padding of each cell in the form layout. The value of the paddingBottom attribute represents the distance from the bottom border of the cell to the bottom of the content of each cell specified in pixels.
- **paddingLeft**
Specifies the left padding of each cell in the form layout. The value of the paddingLeft attribute represents the distance from the left border of the cell to the left side of the content of each cell specified in pixels.
- **paddingRight**
Specifies the right padding of each cell in the form layout. The value of the paddingRight attribute represents the distance from the right border of the cell to the right side of the content of each cell specified in pixels.
- **paddingTop**
Specifies the top padding of each cell in the form layout. The value of the paddingTop attribute represents the distance from the top border of the cell to the top of the content of each cell specified in pixels.
- **valign**
Defines the vertical alignment of the cell content.
 - **BASELINE**
The content of the cell is aligned on the baseline line of the cell (or bottom when no baseline exists).
 - **BOTTOM**
The content of the cell is aligned to the bottom line of the cell.
 - **MIDDLE**
The content of the cell is aligned to the middle of the cell height.
 - **TOP**
The content of the cell is aligned to the top line of the cell.
- **width**
Defines the width of the formLayoutCell. One column can have only one width - when you specify different widths for the same column the width defined last is taken.

Attribute	M	Values	JSP Taglib
align		LEFT (<i>d</i>) RIGHT	<i>Taglib</i> <code>align="LEFT"</code>

Uniform Resource Name (URN)

		CENTER CHAR JUSTIFY	<i>Classlib</i> setHorizontalAlignment (CellHAlign.LEFT)
colspan		Numeric (1)	<i>Taglib</i> colspan="2" <i>Classlib</i> setColspan (2)
content		String or Component	<i>Classlib</i> Text: setContent ("A celltext") Component: setContent (ListBox)
id		String (cs)	<i>Taglib</i> id="Cell01" <i>Classlib</i> setId ("Cell01")
paddingBottom		Numeric (0)	<i>Taglib</i> paddingBottom="1" <i>Classlib</i> setPaddingBottom ("1")
paddingLeft		Numeric (0)	<i>Taglib</i> paddingLeft="5" <i>Classlib</i> setPaddingLeft ("5")
paddingRight		Numeric (0)	<i>Taglib</i> paddingRight="3" <i>Classlib</i> setPaddingRight ("3")
paddingTop		Numeric (0)	<i>Taglib</i> paddingTop="2" <i>Classlib</i> setPaddingTop ("2")
valign		BASELINE BOTTOM MIDDLE (d) TOP	<i>Taglib</i> valign="TOP" <i>Classlib</i> setVerticalAlignment (CellVAlign.TOP)
width		Unit	<i>Taglib</i> width="20%" <i>Classlib</i> setWidth ("20%")

Uniform Resource Name (URN)

Example

Using the taglib

Uniform Resource Name (URN)

```
<hbj:formLayout
  id="myForm"
  marginTop="15px"
  marginRight="30px"
  marginBottom="5px"
  marginLeft="15px"
  width="500px">
  <hbj:formLayoutRow
    id="Row1"
    paddingTop="10px"
    paddingBottom="5px">
    <hbj:formLayoutCell
      id="Cell11"
      align="RIGHT"
      paddingLeft="3"
      paddingTop="5"
      paddingRight="10"
      paddingBottom="5"
      width="40%">
      <hbj:button
        id="myButtonf11"
        text="Button"
      />
    </hbj:formLayoutCell>
    <hbj:formLayoutCell
      id="Cell12"
      align="LEFT"
      paddingLeft="3"
      paddingTop="5"
      paddingRight="10"
      paddingBottom="5">
      <hbj:textView
        text="Celltext aligned left"
      />
    </hbj:formLayoutCell>
  </hbj:formLayoutRow>
  <hbj:formLayoutRow
    id="Row2"
    paddingTop="10px"
    paddingBottom="5px">
    <hbj:formLayoutCell
      id="Cell121"
      align="LEFT"
      paddingLeft="3"
      paddingTop="5"
      paddingRight="10"
      paddingBottom="5">
      <hbj:button
        id="myButtonf21"
        text="Button"
      />
    </hbj:formLayoutCell>
    <hbj:formLayoutCell
      id="Cell122"
      align="RIGHT"
      paddingLeft="3"
      paddingTop="5"
      paddingRight="10"
      paddingBottom="5">
      <hbj:textView
        encode="false"
        text="Celltext aligned right"
      />
    </hbj:formLayoutCell>
  </hbj:formLayoutRow>
</hbj:formLayout>
```

Uniform Resource Name (URN)

Uniform Resource Name (URN)

Using the classlib

Uniform Resource Name (URN)

```
Form form = (Form) this.getForm();
FormLayout fl = new FormLayout();
fl.setId("myForm");

fl.setMarginTop("15px");
fl.setMarginRight("30px");
fl.setMarginBottom("5px");
fl.setMarginLeft("15px");
fl.setWidth("500px");
fl.setDebugMode(true);

FormLayoutRow row1 = fl.addRow();
row1.setPaddingTop("10px");
row1.setPaddingBottom("5px");

Button button = new Button("button", "button");
FormLayoutCell cell11 = fl.addComponent(1,1, button);
cell11.setHorizontalAlignment(CellHAlign.RIGHT);
cell11.setPaddingLeft("3");
cell11.setPaddingTop("5");
cell11.setPaddingRight("10");
cell11.setPaddingBottom("5");
cell11.setWidth("40%");

TextView tv1 = new TextView("tv1");
tv1.setText("Celltext aligned left");

FormLayoutCell cell12 = fl.addComponent(1,2, tv1);
cell12.setHorizontalAlignment(CellHAlign.LEFT);
cell12.setPaddingLeft("3");
cell12.setPaddingTop("5");
cell12.setPaddingRight("10");
cell12.setPaddingBottom("5");
cell12.setWidth("40%");

FormLayoutRow row2 = fl.addRow();
row2.setPaddingTop("10px");
row2.setPaddingBottom("5px");

Button button2 = new Button("button2", "button");
FormLayoutCell cell21 = fl.addComponent(2,1, button2);
cell21.setHorizontalAlignment(CellHAlign.LEFT);
cell21.setPaddingLeft("3");
cell21.setPaddingTop("5");
cell21.setPaddingRight("10");
cell21.setPaddingBottom("5");
cell21.setWidth("40%");

TextView tv2 = new TextView("tv2");
tv2.setText("Celltext aligned right");

FormLayoutCell cell22 = fl.addComponent(2,2, tv2);
cell22.setHorizontalAlignment(CellHAlign.RIGHT);
cell22.setPaddingLeft("3");
cell22.setPaddingTop("5");
cell22.setPaddingRight("10");
cell22.setPaddingBottom("5");
cell22.setWidth("40%");

form.addComponent(fl);
```

Result

<input type="button" value="Button"/>	Celltext aligned left
<input type="button" value="Button"/>	Celltext aligned right

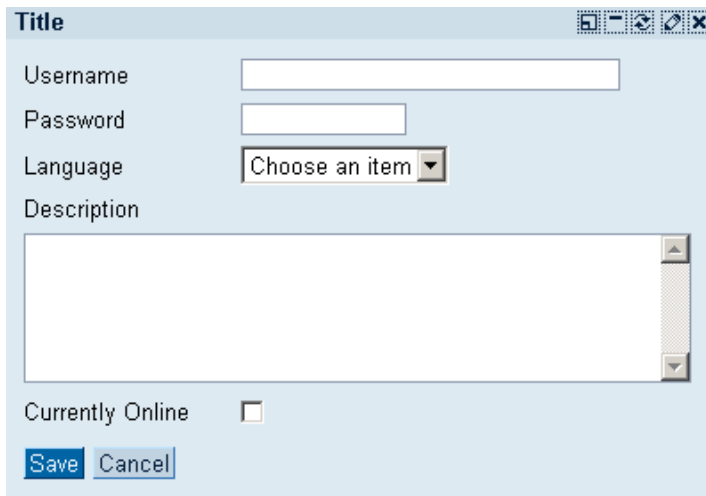
3.4.2.5.6.1 Usage & Type

Definition

The form layout is an invisible control for arranging and aligning controls in an application container, group, or other container in a tabular manner. Elements can also be around wrapped within a cell.

The form layout replaces the previous [grid layout \[Page 33\]](#) control.

Three form layouts allow to arrange the above form elements and buttons in the manner shown; for details see the example



The screenshot shows a form layout titled "Title" with a standard window control bar. The form contains the following elements:

- Username:** A single-line text input field.
- Password:** A single-line text input field.
- Language:** A dropdown menu with the text "Choose an item".
- Description:** A multi-line text area with a vertical scrollbar.
- Currently Online:** A checkbox.
- Buttons:** "Save" and "Cancel" buttons at the bottom left.

Use

Use the form layout to align controls within containers in a tabular fashion. Especially, use it in groups, tabstrips and trays (iViews). The grid defined by the form layout is composed of rows, which contain cells. Thus, rows and columns of the grid are defined implicitly. Cells can span multiple columns. In addition, elements in a cell can wrap around. Various controls can be added to the cells.

You can nest form layouts for arranging page elements on different levels.

When use the Form Layout

- Use the form layout to align controls with respect to other controls in the user interface. The most common usage of the form layout is: (1) laying out forms inside containers, and (2) arranging different containers or form layouts. See the example below for details and both uses.
- You can also use the form layout if you do not need to align controls with other elements in your interface. In this case, insert only one cell into the respective rows and set the cell's width so that it exceeds the width of the form layout.



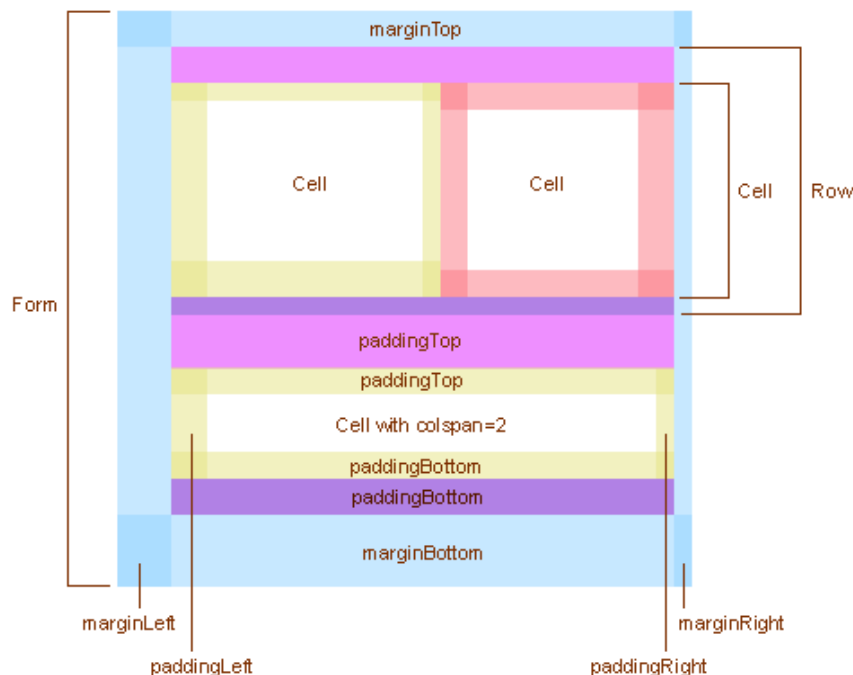
The form layout is similar to the [grid layout \[Page 33\]](#) but has more features for adjusting the grid cells. You need not specify a fixed number of rows and columns but simply add rows and cells within rows. The form layout also can wrap around elements in a cell but the [flow layout \[Page 33\]](#) is more efficient for this purpose.

Overview of the Elements and Spacing

The form layout consists of three elements, each of which has a spacing of its own:

- **Form:** `marginBottom`, `marginLeft`, `marginRight`, `marginTop` define the spacing in pixels between the border of the form layout and its content area; each margin is set to zero by default.
- **Row:** `paddingBottom`, `paddingTop` define the padding in pixels at the top and bottom of a row; set to zero by default.
- **Cell:** `paddingBottom`, `paddingLeft`, `paddingRight`, `paddingTop` define the spacing in pixels between the border of the cell and its content area; set to zero by default.

Overview of the different margins and paddings within a form layout:



Uniform Resource Name (URN)

When adding elements to a form layout, take care that the margins and the padding at the different levels are set to the correct values. When nesting form layouts, the margins at the form level have to be left at the default value of zero in order to avoid additional padding. See the example below for details.

Example

The following example demonstrates how to use the form layout. First, identify the different areas within your application that need to be aligned. In the example in figure 3 there are two areas, a form area containing labels and input elements, and a button row. Both areas are aligned using separate form layouts. The form layout at the top contains the form area, that is, the labels and the input elements. The second form layout includes the *Save* and *Cancel* buttons. Then both form layouts are arranged using a third form layout.

Top Form Layout

First create a form layout for the input elements. Set all margins (**marginBottom**, **marginLeft**, **marginRight**, **marginTop**) at the form level to 5 pixels. Note that the rules for [Spacing Between Grouped Controls \[Page 33\]](#) require a spacing of 10 pixels between the tray border and its content. As 5 pixels are already provided by the tray, the form layout has to provide the remaining 5 pixels at the appropriate borders.

Then add six rows to the top form layout and add two cells to each row except the fifth one for achieving a two column layout. In the fifth row, add only one cell and set `colspan=2`.

The rules in [Spacing Between Single Controls \[Page 33\]](#) require a spaces of 5 pixels between rows of screen elements. As the top and bottom spacings are already set correctly, set the bottom padding (**paddingBottom**) to 5 pixels for all but the last row.



There are alternatives for achieving a 5 pixels spacing between rows but this approach seems to be the easiest way to do it.

For achieving the correct horizontal spacing between the labels and their corresponding input elements, do not specify a value for the width of the left cells. Specify a right padding (**paddingRight**) between 8 and 22 pixels for the left cells, instead. Leave all other cell paddings at their default values of zero.

Finally, add the labels and input elements to the respective cells.

Bottom Form Layout

Create a second form layout for the buttons and set all margins at the form level to 5 pixels. As there are 5 pixels at the bottom of the top form layout and 5 levels at the top of this form layout, you get automatically a spacing of 10 pixels between the bottom input element and the buttons.

Add only one row and two columns for the two buttons to the second form layout. Then add a button to each cell.

Leave the padding at the row and cell levels at their default values of zero, except for the right cell padding of the left cell. Set this padding (**paddingRight**) to 5 pixels. This ensures the correct horizontal spacing between the two buttons.

Two form layouts are used for arranging the controls

Uniform Resource Name (URN)

Form Layout for the form with margins of 5px

Row containing two cells - row borders not shown
Rowpadding paddingBottom set to 5 pixels, except for the bottom row

Form Layout for the button row with margins of 5px
Right padding of left cell set to 5 pixels

Arranging the Form Layouts

As the two form layouts are to be appear below each other, you need a third form layout that includes both. This outer form layout needs two rows and one cell in each row, resulting in only one column (see figure 4). Simply add the two form layouts to the cells.

Note that this outer form layout must not have additional padding or margins. Therefore, let all four margins at the form level at their default values of zero. Also, leave all other padding values at their defaults.

A third form layout is used for arranging the two areas vertically

Form Layout to separate form and buttons - margins set to zero



You can temporarily set the attribute **debugMode** to TRUE to render the form layout with frames. This makes it easier for you to achieve a proper layout (see [Control API for Form Layout \[Page 33\]](#) for details).

Design-relevant Attributes

The form layout has design-relevant attributes on the form level, the row level, and the cell level. There are also some dependencies between the attribute values on the different levels. See figure for an overview of the form layout, its elements and spacings.

Form Level

- **marginBottom, marginLeft, marginRight, marginTop:** Define the spacing in pixels between the border of the form layout and its content area; each is set to zero by default.
- **width:** The width can be specified in pixels or percent of the including container width. If the width of a cell is also specified in percent and it exceeds the form layout's width the cell content will be wrapped.

Row Level

- **paddingBottom, paddingTop:** Define the padding in pixels at the top and bottom of a row; set to zero by default.

Cell Level

- **align** (LEFT, RIGHT, CENTER, CHAR, JUSTIFY): Defines the horizontal alignment of elements within a cell.
- **valign** (BASELINE, BOTTOM, MIDDLE, TOP): Defines the vertical alignment of elements within a cell.
- **paddingBottom, paddingLeft, paddingRight, paddingTop:** Define the spacing in pixels between the border of the cell and its content area; set to zero by default.
- **Width:** Defines the width in pixels or percent of the form layout. Note that if different widths are specified for a column the last value is used in order to avoid conflicts. Also note that a value exceeding the width of the form layout will cause wrapping behavior.
- **colSpan:** Defines the horizontal expansion of a cell in columns.

You can also use the Boolean attribute **debugMode** as an aid for achieving a proper layout. If it is set to TRUE the cell borders are displayed.

Further Information

[Browser Support & Accessibility \(508\) \[Page 33\]](#)

[Control API \[Page 33\]](#)

Related Controls

[Flow Layout \[Page 33\]](#)

[Grid Layout \[Page 33\]](#)

3.4.2.5.6.2 Browser Support & 508

The form layout control structures elements in every browser.

Editability in Style Editor

Currently, the form layout is not changeable by the Style Editor.

Accessibility - 508 Support

The form layout has no special accessibility enhancements. It can contain several controls that are by themselves in the accessible hierarchy and might have further descriptions for blind users.

3.4.2.5.7 Grid Layout

Control API

A grid is a two dimensional arrangement of data in rows and columns. To avoid unexpected results, the rows and columns should always be defined by `gridLayoutCells` instead of using the `'columnSize'` and `'rowSize'` attribute. With the `gridLayoutCell` you have control over the width of the `gridLayoutCell` while using the `"columnSize"` the renderer and web client take the control. Especially in combination with `'debugMode'` attribute set to `"FALSE"`, the layout of the grid is not displayed as expected.

If no `gridLayoutCells` are defined no `gridLayout` is displayed.

Limitation:

Large `gridLayouts` (large amount of rows and columns) can cause problems like:

- Compiler errors that are caused by a 64kB method length limit.
- Slow processing of page because of huge HTML-Code generated by the JSP-Compiler

To avoid these problems you could use the `gridLayout` tag and combine it with `<tr>` & `<td>` tags.

- **cellPadding**

Defines the padding of each cell in the grid layout. The value of the cell padding attribute represents the distance from the border of the cell to the content of each cell specified in pixels.



The `cellPadding` is applied to the top, left, right and bottom of the cell.

- **cellSpacing**

Specifies the space between the left side of the `gridLayout` and the left-hand side of the leftmost `gridLayoutCell`, the top of the `gridLayout` and the top side of the topmost row and so on for the right and bottom of the `gridLayout`. The attribute also specifies the amount of space to leave between the `gridLayoutCells`.

Defines the spacing between cells and the outer boundary in the grid layout in pixels.

- **columnSize**

Defines the number of columns for the `gridLayout`. The columns are defined with the `gridLayoutCell` control and `'ColumnSize'` is overruled by the `gridLayoutCell` definition.

- **debugMode**

A Boolean value. If set to `"TRUE"` the `gridLayoutCell` is rendered with a frame. The frame size is defined by `gridLayoutCell` `'width'` and the `'cellpadding'`. If a `gridLayoutCell` is not defined or empty no frame is displayed.

If set to `"FALSE"` no frame is rendered. Please check the `gridLayout` description above for limitations.

Uniform Resource Name (URN)

By default the borders of the grid are invisible. To see the borders for layout and debug reasons set the debug attribute.



Setting the debugMode attribute will add pixels to visualize borders. Therefore the sizes of the grid layout will change if you reset the attribute. The debugMode attribute, as indicated by the name, should only be used for debugging and not for "styling".

- **heightPercentage**
Sets the height of the grid in percent.
- **id**
Identification name of the gridLayout.
- **rowSize**
Defines the number of rows for the gridLayout. The 'rowSize' is overruled when more rows are defined with the gridLayoutCell control then specified with the 'rowSize' attribute. If 'rowSize' is higher than the rows defined by the gridLayoutCell, the frame height of the gridLayout is extended.
- **width**
Defines the width of the gridLayout. If the 'width' in gridLayoutCell is specified in percent, the percentage will be calculated from the width of the gridLayout and not from the width of the form.
If the gridLayoutCell definition exceeds the 'width' of the gridLayout the gridLayoutCell content will be wrapped.

Attribute	M	Values	Usage
cellPadding		Numeric (0)	<i>Taglib</i> cellPadding="5" <i>Classlib</i> setCellPadding(5)
cellSpacing		Numeric (0)	<i>Taglib</i> cellSpacing="5" <i>Classlib</i> setCellSpacing(5)

Uniform Resource Name (URN)

columnSize		Numeric (0)	<i>Taglib</i> columnSize="3" <i>Classlib</i> setColumnSize(3)
debugMode		FALSE (d) TRUE	<i>Taglib</i> debugMode="TRUE" <i>Classlib</i> setDebugMode(true)
heightPercentage		Numeric (0)	<i>Classlib</i> setHeightPercentage(20)
id	*	String (cs)	<i>Taglib</i> id="ZIPCode_grid" <i>Classlib</i> setId("ZIPCode_grid")
rowSize		Numeric	<i>Taglib</i> rowSize="5" <i>Classlib</i> setRowSize(5)
width		Unit (100%)	<i>Taglib</i> width="500" <i>Classlib</i> setWidth("500")

gridLayoutCell

Defines the cells in the gridLayout.

- **columnIndex**
Defines the horizontal position of the cell.
- **colSpan**
Defines the horizontal expansion the cell in percent. If you specify for example, 100, the cell uses the whole gridLayout width. Cells right of this cell are omitted. A possible application for this attribute is to display headlines in the gridLayout.
- **content**
Specifies the content for the cell.
- **heightPercentage**
Sets the height of the gridLayout cell in percent.
- **horizontalAlignment**
Defines the horizontal alignment of the cell content.
 - LEFT
Left justifies the content of the cell.
 - RIGHT
Right justifies the content of the cell.
 - CENTER
Centers the content of the cell.
 - CHAR
Aligns text around a specific character. Not supported by all web clients.

Uniform Resource Name (URN)

- JUSTIFY
Sets text in the cell left and right aligned. Not supported by all web clients.
- **id**
Identification name of the GridLayoutCell.
- **rowIndex**
Defines the vertical position of the cell.
- **style**
Defines the stylesheet to be used to display the cell.
- **verticalAlignment**
Defines the vertical alignment of the cell content.
 - BASELINE
The content of the cell is aligned on the baseline line of the cell (or bottom when no baseline exists).
 - BOTTOM
The content of the cell is aligned to the bottom line of the cell.
 - MIDDLE
The content of the cell is aligned to the middle of the cell height.
 - TOP
The content of the cell is aligned to the top line of the cell.
- **width**
Defines the width of the GridLayoutCell. One column can have only one width - when you specify different widths for the same column the width defined last is taken.

Attribute	M	Values	Usage
columnIndex	*	Numeric	<i>Taglib</i> <code>columnIndex="2"</code> <i>Classlib</i> <code>setColumnIndex(2)</code>
colSpan		Numeric (0)	<code>setColSpan(100)</code>
content		String	<i>Classlib</i> Text: <code>setContent("Celltext")</code> Component: <code>setContent(ListBox)</code>
heightPercentage		Numeric (0)	<i>Classlib</i> <code>setHeightPercentage(20)</code>
horizontalAlignment		LEFT (d) RIGHT CENTER CHAR JUSTIFY	<i>Taglib</i> <code>horizontalAlignment="LEFT"</code> <i>Classlib</i> <code>setHAlignment(CellHAlign.LEFT)</code>
id		String (cs)	<i>Taglib</i> <code>id="Cell101"</code> <i>Classlib</i> <code>setId("Cell101")</code>
rowIndex	*	Numeric	<i>Taglib</i>

Uniform Resource Name (URN)

			<code>rowIndex="1"</code> <i>Classlib</i> <code>setRowIndex(1)</code>
style		String	<i>Taglib</i> <code>style="WildStyle"</code> <i>Classlib</i> <code>setStyle("WildStyle")</code>
verticalAlignment		BASELINE BOTTOM MIDDLE (<i>d</i>) TOP	<i>Taglib</i> <code>verticalAlignment="TOP"</code> <i>Classlib</i> <code>setVAlignment(CellVAlign.TOP)</code>
width		Unit	<i>Taglib</i> <code>width="20%"</code> <i>Classlib</i> <code>setWidth("20%")</code>

Example*Using the taglib*

Uniform Resource Name (URN)

```
<hbj:gridLayout
  id="myGridLayout1"
  debugMode="True"
  width="40%"
  cellSpacing="5">
  <hbj:gridLayoutCell
    rowIndex="1"
    columnIndex="1"
    width="10%"
    horizontalAlignment="RIGHT">
    <hbj:button
      id="myButton1"
      text="Button"
      tooltip="Button in the 1st row"
    />
  </hbj:gridLayoutCell>
  <hbj:gridLayoutCell
    id="myGridLayoutCell2"
    rowIndex="1"
    columnIndex="2"
    width="40%"
    horizontalAlignment="LEFT"
    verticalAlignment="BOTTOM">
    Celltext aligned left
  </hbj:gridLayoutCell>
  <hbj:gridLayoutCell
    rowIndex="2"
    columnIndex="1"
    width="20%">
    <hbj:button
      id="myButton2"
      text="Button"
      tooltip="Button in the 2nd row"
    />
  </hbj:gridLayoutCell>
  <hbj:gridLayoutCell
    rowIndex="2"
    columnIndex="2"
    horizontalAlignment="RIGHT">
    Celltext aligned right
  </hbj:gridLayoutCell>
</hbj:gridLayout>
```

Uniform Resource Name (URN)

Using the classlib

```

Form form = (Form) this.getForm();
GridLayout gl = new GridLayout();
gl.setId("myGrid");

gl.setCellSpacing(5);
gl.setWidth("40%");
gl.setDebugMode(true);

Button button = new Button("button", "button");
GridLayoutCell cell11 = new GridLayoutCell("cell11");
cell11.setHAlignment(CellHAlign.RIGHT);
cell11.setWidth("10%");
cell11.setContent(button);
gl.addCell(1, 1, cell11);

TextView tv1 = new TextView("tv1");
tv1.setText("Celltext aligned left");

GridLayoutCell cell12 = new GridLayoutCell("cell12");
cell12.setHAlignment(CellHAlign.LEFT);
cell12.setWidth("40%");
cell12.setContent(tv1);
gl.addCell(1, 2, cell12);

Button button2 = new Button("button2", "button");
GridLayoutCell cell21 = new GridLayoutCell("cell21");
cell21.setHAlignment(CellHAlign.LEFT);
cell21.setWidth("10%");
cell21.setContent(button2);
gl.addCell(2, 1, cell21);

TextView tv2 = new TextView("tv2");
tv2.setText("Celltext aligned right");

GridLayoutCell cell22 = new GridLayoutCell("cell22");
cell22.setHAlignment(CellHAlign.RIGHT);
cell22.setWidth("40%");
cell22.setContent(tv2);
gl.addCell(2, 2, cell22);

form.addComponent(gl);

```

Result

<input type="button" value="Button"/>	Celltext aligned left
<input type="button" value="Button"/>	Celltext aligned right

3.4.2.5.7.1 Usage & Type

Definition

The grid layout is an invisible control that helps you in arranging and aligning controls in an application, group or other container in a tabular manner.

Grid layout arrangement of controls

Use



The grid layout is obsolete - use the [form layout \[Page 33\]](#), instead.

Currently, it is not possible to achieve the exact paddings and spacings as recommended in the "Layout" section. The suggestions given here are approximations of the optimal Use the grid layout to align controls within containers in a tabular fashion. The grid consists of cells that are arranged in rows and columns. Various controls can be added to the cells. You can insert the grid layout into any container control. Especially, use it in groups, tabstrips and trays (iViews). You can also nest grid layouts for arranging page elements on different levels (see example below).

The most common usage of the grid layout is the layout of forms inside containers and the arrangement of different containers. There are two attributes for managing the spacing between rows and columns, **cellSpacing** and **cellPadding**. For both of the above cases a **cellSpacing** of 5 is recommended. There is no need to use **cellPadding** for these default layouts. See the example below for details.



Currently, it is not possible to achieve the exact paddings and spacings as recommended in the "Layout" section. The suggestions given here are approximations of the optimal layout. Use these values until new controls are introduced, which can deal with the layout issues of the grid and flow layout controls.

When Use the Grid Layout - When Use the Flow Layout

- Use the grid layout to align controls with respect to other controls in the user interface

Uniform Resource Name (URN)



Use the [form layout \[Page 33\]](#) instead of the grid layout

- Use the flow layout if you do not need to align controls with other elements in your interface; this will enhance performance because the flow layout does not have an overhead of table structures in the rendering

Example

The following example demonstrates how to use the grid layout. First, identify the different areas within your application that need to be aligned. In the example there are two areas, a form area containing the input elements that have to be aligned, and a button row, where buttons have to be added.

Both areas consist of separate grid layouts. The grid at the top contains the form area, the labels, and the input elements. The second grid includes the *Save* and *Cancel* buttons. As the spacing between the controls should always be 10px, set the **cellSpacing** to 5.

Step 1 - grid layout arrangement of inner controls

Grid Layout for the form with cell spacing of 5px

Grid Layout for the button row with cellspacing of 5px

These two areas should appear below each other. Therefore, you need another grid consisting of one column only and two rows. In the cells you simply add the form area and the button row (see figure 3). This grid does not need additional padding or spacing because the two added areas already have the correct spacing.

Step 2 - grid layout arrangement of areas

Grid Layout to separate form and buttons



You can temporarily set the attribute **debugMode** to TRUE to render the grid layout with frames. This helps you in achieving a proper layout (see [Control API for Grid Layout \[Page 33\]](#) for details).

Design-relevant Attributes

You can set the number of columns (**columnSize**), and the spacing within (**cellPadding**) and between cells (**cellSpacing**).

You can also use the Boolean attribute **debugMode** as an aid for achieving a proper layout. If it is set to TRUE the cell borders are displayed.

Further Information

[Browser Support & Accessibility \(508\) \[Page 33\]](#)

[Control API for Grid Layout \[Page 33\]](#)

Related Controls

[Flow Layout \[Page 33\]](#)

[Form Layout \[Page 33\]](#)

3.4.2.5.7.2 Browser Support & 508

The grid layout control structures elements in every browser.

Editability in Style Editor

Currently, the grid layout is not changeable by the Style Editor.

Accessibility - 508 Support

The grid layout has no special accessibility enhancements. It can contain several controls that are by themselves in the accessible hierarchy and might have further descriptions for blind users.

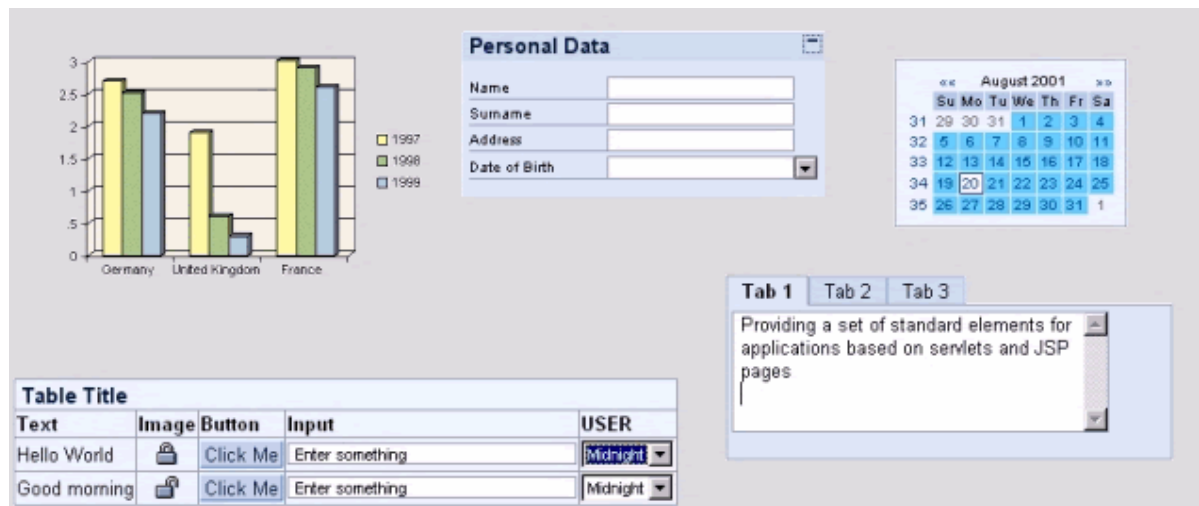
3.4.2.6 Visible Controls

Purpose

GUI elements that are used to build an application. The controls are placed in a form. Every control has different attributes that define the "look" of the control. Controls are checkboxes, radio buttons and grids to name a few.

HTMLB controls example: Chart, date navigator, tabstrip and table view.

Uniform Resource Name (URN)



Visible Controls

Breadcrumb [Page 33]	Drop Target [Page 33]	Item List [Page 33]	Scroll Container [Page 33]
Button [Page 33]	File Upload [Page 33]	Label [Page 33]	Table View [Page 33]
Button Row [Page 33]	Group [Page 33]	Link [Page 33]	Tabstrip [Page 33]
Chart [Page 33]	Hover Menu [Page 33]	List Box [Page 33]	Text Edit [Page 33]
Checkbox [Page 33]	HTML Edit [Page 33]	Menu Bar [Page 33]	Text View [Page 33]
Date Navigator [Page 33]	Image [Page 33]	Nonisolated HTML Container [Page 33]	Tool Bar [Page 33]
Drag Source [Page 33]	Input Field [Page 33]	Progress Indicator [Page 33]	Tree View [Page 33]
Dropdown List Box [Page 33]	Isolated HTML Container [Page 33]	Radio Button [Page 33]	

3.4.2.6.1 Breadcrumb

Definition

The breadcrumb represents the sequence of the visited pages (remember the story of Hansel + Gretel). It informs the user about his actual position in your application and allows easy navigation back to start page. An item in the breadcrumb chain is called breadcrumbItem. BreadcrumbItems can be defined with models or manually.

Uniform Resource Name (URN)

If the breadcrumb line becomes longer than the web client window it is word wrapped like a text line - if there is no word separator in the breadcrumbItem value string, the line is not wrapped.

- **behavior**
The breadcrumb can behave as a
 - **DEFAULT**
Each breadcrumbItem can be linked independently.
 - **SINGLELINK**
The entire breadcrumb string is a single link.
- **id**
Identification name of the breadcrumb.
- **model**
Defines the model or bean which provides the breadcrumb with data.
See also [ListModel \[Page 33\]](#).
- **nameOfKeyColumn**
Specifies the name of the column that contains the key. This is used when you use an underlying table in the model.
- **onClick**
Defines the event handling method that will be processed when the user clicks on the breadcrumb. The `BreadCrumbEventClick` object allows access to the key, which breadcrumb element had been clicked and allows the definition of parameters.
- **size**
Defines the text size of the breadcrumb. Possible values are:
 - **LARGE**
Double the standard text size.
 - **MEDIUM**
Standard text size.
 - **SMALL**
Half of the standard text size.
- **tooltip**
Defines the hint of the button which is displayed as the mouse cursor passes over the button, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
behavior		DEFAULT (d) SINGLELINK	<i>Taglib</i> <code>behavior="SINGLELINK"</code> <i>Classlib</i> <code>setBehavior(BreadCrumbBehavior. SINGLELINK)</code>
id	*	String (cs)	<i>Taglib</i> <code>id="OrderConfirm"</code> <i>Classlib</i> <code>setId("OrderConfirm")</code>
model		String (cs)	<i>Taglib</i> <code>model="bean.model [Page 33]"</code> <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setModel ((IListModel [Page 33]) model)</code>
name OfKey Column		String	<i>Taglib</i> <code>nameOfKeyColumn="col1"</code> <i>Classlib</i> <code>setNameOfKeyColumn("col1")</code>
tooltip		String	<i>Taglib</i> <code>tooltip="Confirm order"</code> <i>Classlib</i> <code>setTooltip("Confirm order")</code>
size		LARGE MEDIUM SMALL	<i>Taglib</i> <code>size="MEDIUM"</code> <i>Classlib</i> <code>setSize(BreadCrumbSize.MEDIUM)</code>

Events	M	Values	Usage
onClick		String (cs)	<i>Taglib</i> <code>onClick="ProcessCrumb"</code> <i>Classlib</i> <code>setOnClick("ProcessCrumb")</code>

breadCrumbItem

Defines the items in the breadCrumb instead of the model.



We strongly recommend models to supply the breadCrumb with data.

- **key**
A string which is passed on to the event handling routine when the event occurs. A key string has to be defined and must not be empty.
If the attribute 'behavior' is set to "SINGLELINK" the 'key' is set to "null" when passed on to the event handling routine.
- **value**
Defines the text string displayed in the breadCrumb. A value string has to be defined and must not be empty.

Attributes	M	Values	Usage
key	*	String (cs)	<i>Taglib</i> <code>key="EVK1"</code> <i>Classlib</i> <code>addItem ("EVK1","1stVisitedPage")</code>
value	*	String	<i>Taglib</i> <code>value="1stVisitedPage"</code> <i>Classlib</i> <code>addItem ("EVK1","1stVisitedPage")</code>

BreadCrumbClickEvent

Allows access to the event fired by the breadCrumb control.

- **setParams**
Set the parameters for the HTML page. The parameters depend on the nature of the event. For example, table related events can include row and column id. The `setParams` method is inherited from the `com.sapportals.htmlb.event.Event` class. See the HTMLB Javadoc - Class "Event" for more details on how to set parameters for the HTML page and how to have access.
- **getKey**
Returns the key of the breadCrumb control element, that has been clicked.
-

Example

using the taglib

```
<hbj:breadcrumb
  id="myNavigation"
  tooltip="Navigation and orientation in the application"
  onClick="ProcessbreadcrumbClick"
  size="SMALL"
>
  <hbj:breadcrumbItem key="EVK1" value="MainLevel" />
  <hbj:breadcrumbItem key="EVK2" value="1stLevel" />
  <hbj:breadcrumbItem key="EVK3" value="2ndLevel" />
  <hbj:breadcrumbItem key="EVK4" value="3rdLevel" />

</hbj:breadcrumb>
```

using the classlib

```
Form form = (Form) this.getForm();
Breadcrumb bc = new Breadcrumb("myNavigation");
bc.addItem("EVK1", "MainLevel");
bc.addItem("EVK2", "1stLevel");
bc.addItem("EVK3", "2ndLevel");
bc.addItem("EVK4", "3rdLevel");
bc.setSize(BreadcrumbSize.MEDIUM);
bc.setTooltip("Navigation and orientation in the application");
bc.setOnClick("ProcessbreadcrumbClick");
form.addComponent(bc);
```

Result

[MainLevel](#) > [1stLevel](#) > [2ndLevel](#) > [3rdLevel](#)

3.4.2.6.1.1 Usage & Type

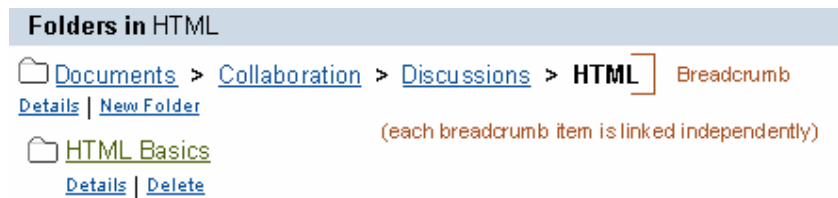


Figure 1: Search result using breadcrumbs

Breadcrumbs can be used for:

- inform users about their current position within a hierarchy, such as an application, a directory, a Website, or a document.
- Allow for easy navigation back to the starting point, or to other levels within a hierarchy

Usage

Software applications and information in the Web are often organized hierarchically: General information may lead to more specific information, thus creating an information hierarchy. The breadcrumb control informs the user about the path to a specific content within such a hierarchy. For example, hit lists typically include a breadcrumb to inform users about the hierarchy level of a search result and therefore are guides to the list items. If the breadcrumb uses links in the path description, the user can move to a specific folder or topic.

An item in the breadcrumb chain is called breadcrumb item. Breadcrumb items can be defined by models or manually.



Figure 2: A wrapped breadcrumb

If the breadcrumb line becomes longer than the width of the browser window, the breadcrumb is word-wrapped like a text line (figure 2). The wrapping is done at word separators, such as blanks. If there is no word separator in the breadcrumb item string, the breadcrumb will wrap behind the breadcrumb item separator ">".

Types

Breadcrumbs can be displayed as simple path information (no link, figure 3 top), as a chain of clickable locations within the hierarchy (figure 3 center), or as one link that is described by the path information (figure 3 bottom).

FirstPathItem > SecondPathItem > ThirdPathItem

Uniform Resource Name (URN)

[FirstPathItem](#) > [SecondPathItem](#) > **ThirdPathItem**

[FirstPathItem](#) > [SecondPathItem](#) > [ThirdPathItem](#)

Figure 3: Simple path information (top), each breadcrumb item is linked independently (center), whole path is selectable (bottom).

The breadcrumb type is set through the attribute behavior: value SINGLELINK creates a breadcrumb, where the whole path is selectable; value DEFAULT creates a breadcrumb, where each item can be linked independently.

Usage - Types

Use the different breadcrumb types for the following purposes:

- Use path information breadcrumbs (figure 3 top) for indicating the location of files inside a hierarchy.
Example: A list of search results not only shows the hits themselves but also their paths.
- Use independently linked breadcrumbs (figure 3 center) if you want to allow users to move up and down within a hierarchy, or to jump to a certain category. The last breadcrumb always shows the actual page and is no link.
- Use the whole breadcrumb path as one single link (figure 3 bottom) to inform users about the location of a link target (= the last breadcrumb item) inside a hierarchy.

Design-relevant Attributes

size

Breadcrumbs come in three different font sizes: large, medium (= default) and small (figure 4). Set the attribute size to the values LARGE, MEDIUM, or SMALL.

FirstPathItem > SecondPathItem > ThirdPathItem

FirstPathItem > SecondPathItem > ThirdPathItem

FirstPathItem > SecondPathItem > ThirdPathItem

Figure 4: Large, medium, and small size breadcrumbs

Use the text size that correspond to the size of the surrounding text. In case space requirements are tight, use smaller text sizes if available. These sizes may also be used for design and highlighting reasons.

Releated Controls

[Links \[Page 33\]](#)

3.4.2.6.1.2 Browser Support & 508

In Netscape 4.x and 6.x the breadcrumb path icons are not bold but have normal font weight.

Netscape path icons have normal font weight

[Text 1 > Text 2 > Text 3](#)

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the tree view control:

Group	Style	IE 5 and above	Netscape 4.7
Breadcrumb Styles	Font Color of Breadcrumb Path Icon	X	X
	Font Weight of Breadcrumb Path Icon	X	X
	Text Decoration of Active Entry	X	X
	Breadcrumb Padding	X	

For common styles see section [HTMLB Controls and Style Editor \[Page 33\]](#).

Accessibility - 508 Support

- **Keyboard:** The breadcrumb control is inserted into the accessibility hierarchy by default if it contains links.
- **Default Description:** Is provided by the HTMLB rendering engine.
- **Application-specific Description:** Set an additional description using the *setTooltip* method if needed.

3.4.2.6.2 Button

Definition

Provides any type of functionality in your application at the touch of the button. Hints can be displayed as the mouse cursor passes over the button, or as the mouse button is pressed but not released.

- **design**
Defines the size and highlighting of the button.

Uniform Resource Name (URN)

- STANDARD

Displays the button with the standard background and text color.

- SMALL

Displays the button with the standard background and text color and half of the STANDARD size.

- EMPHASIZED

Displays the button with the highlighted background and text color. You can also refer to the emphasized button as default button. Therefore only one emphasized button per form can be defined. If you use more than one "EMPHASIZED" button, the last button defined becomes "EMPHASIZED".

- **disabled**

A boolean value that defines if the button is clickable. If the button is disabled it sends no event when you press a mouse button on the button. A disabled button has a different text color to show the user that it is disabled.

- **encode**

A Boolean value that defines how the text in the button is interpreted. HTML text formatting commands (for example, <h1>, <i> etc.) can be used to change the display of the text. If there are no formatting commands in the text string, the encode attribute has no effect.

Example:

```
text = "<h1><i>Important</i></h1>"
```

```
encode = "false"
```

Browser output:

Important

the text string is rendered by interpreting the formatting commands.

```
encode = "true"
```

Browser output: <h1><i>Important</i></h1>

the formatting commands are displayed and not interpreted.

- **id**

Identification name of the button.

- **onClick**

Defines the event handling method that will be processed when the user clicks on the enabled button. If you do not define a 'onClick' event the button can be clicked but no event is generated.

- **onClientClick**

Defines the JavaScript fragment that is executed when the user clicks on the button. If both events ('onClick' and 'onClientClick') are specified, the 'onClientClick' event handling method is activated first. By default the 'onClick' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onClick' event handling method with the command:

```
htmlbevent.cancelSubmit=true;
```

The 'onClientClick' event is useful to preprocess the form and only send the form to client if the preprocessing was successful (e.g. date validation, valid number format etc.) to save client/server interaction.

Uniform Resource Name (URN)



A button click usually activates the client/server interaction. If an input field has to be filled out for further processing, the JavaScript fragment can check the necessary input on the client side and display a message if the necessary input is missing, without server interaction.



To use JavaScript the JSP has to use the page tag (set page tag).

- **text**

Defines the string of text placed centered on the button. If no text should be displayed in the button an empty string (null) can be used. The width of the button is automatically adjusted to the length of the text.

- **width**

Defines the width of the button. The width of the button is automatically adjusted to the length of the 'text'. To see an effect of the 'width' attribute, 'width' has to be set higher as the width defined through the length of the 'text' string. The text string of the button is always placed centered on the button. If an empty (null) 'text' string is set no 'text' attribute is defined the width of the button is set according to the 'width' attribute.

- **tooltip**

Defines the hint of the button which is displayed as the mouse cursor passes over the button, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
design		STANDARD (d) SMALL EMPHASIZED	<i>Taglib</i> design="STANDARD" <i>Classlib</i> setDesign (ButtonDesign.STANDARD)
disabled		FALSE (d) TRUE	<i>Taglib</i> disabled="FALSE" <i>Classlib</i> setDisabled (true)
encode		FALSE (d) TRUE	<i>Taglib</i> not available <i>Classlib</i> setEncode (true)
id	*	String (cs)	<i>Taglib</i> id="OrderConfirm" <i>Classlib</i> setId ("OrderConfirm")
text		String	<i>Taglib</i> text="Confirm" <i>Classlib</i> setText ("Confirm")

Uniform Resource Name (URN)

width		Unit	<i>Taglib</i> width="125px" <i>Classlib</i> setWidth("125px")
tooltip		String	<i>Taglib</i> tooltip="Confirm order" <i>Classlib</i> setTooltip("Confirm order")

Events	M	Values	Usage
onClick		String (cs)	<i>Taglib</i> onClick="ProcessConfirm" <i>Classlib</i> setOnClick("ProcessConfirm")
onClientClick		String (cs)	<i>Taglib</i> onClientClick="alert('Hi');" <i>Classlib</i> setOnClientClick("alert('Hi');")

Example

using the taglib

```
<hbj:button
  id="OrderConfirm"
  text="Confirm"
  width="125px"
  tooltip="Click here to confirm order"
  onClick="ProcessConfirm"
  disabled="false"
  design="STANDARD"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
Button button = new Button("button", "button");
button.setText("Confirm");
button.setWidth("125px");
button.setTooltip("Click here to confirm order");
button.setOnClick("ProcessConfirm");
button.setDisabled(false);
button.setDesign(ButtonDesign.STANDARD);
form.addComponent(button);
```

Result



3.4.2.6.2.1 Usage & Type

Buttons are used for explicit functions that refer to a given object or serve for navigational purposes.

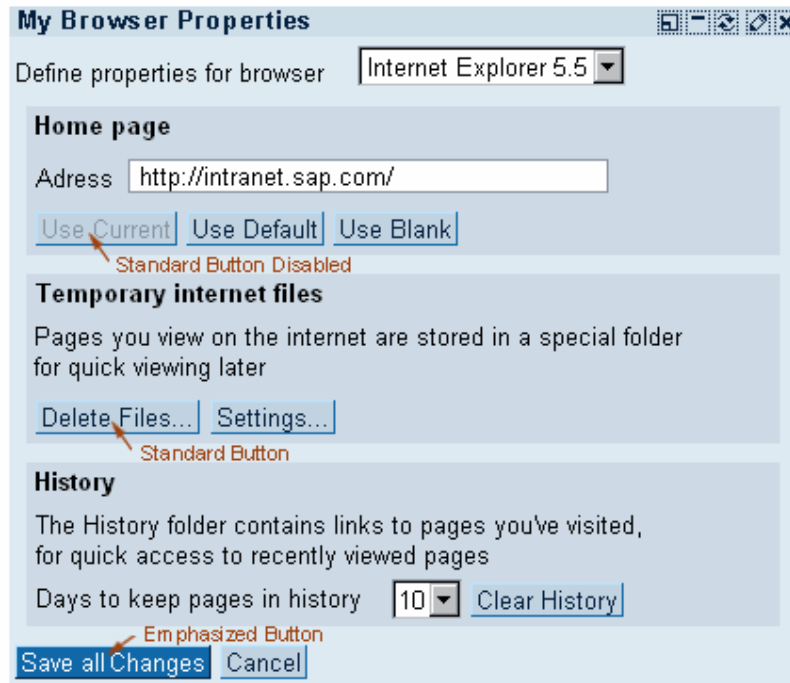


Figure 1: Example of an iView containing groups with buttons and two buttons belonging to the iView itself

Usage

Use buttons only for few and very important functions. A lot of buttons make a screen look heavy and complex. Buttons have optical weight and visual dominance. Therefore, when in doubt about whether to use a link or a button, go with a link.



For a detailed discussion of when to use buttons and when to use links, see [Link \[Page 33\]](#).

Pressing the Enter key activates the default function (mostly, but not necessarily, identical to the emphasized button's function).

Usage guidelines for the different button types and sizes are presented below.

Labeling

Use title case for button labels. Use the ellipsis character ("...") on button labels to indicate that the command needs further information to execute. Typically, the user is presented a dialog to fill in missing information.



Title case means that the first letter of each word is capitalized, except for certain small words, such as articles and short prepositions.

Choose the button's function description carefully; try to be as explicit as possible. For complex interactions, use verb-noun combinations, e.g. "Search Database". If the context is clear, i.e. if the action can only be applied to one object, it is sufficient to use a single verb as the button's label ("Search"). For shufflers and comparable elements, you can also use a simple "Go".

Positioning and Design Alternatives

For detailed button positioning rules see Button Positioning.

Buttons are similar in function to links. For a discussion of when to use buttons and when to use links, see [Link \[Page 33\]](#).

Types

HTMLB offers three different button types (attribute design, values STANDARD, SMALL, EMPHASIZED):



Figure 2: Standard, small, and emphasized button.

In the following, we list usage guidelines for these types.

Usage - Emphasized Buttons vs. Standard Button

For functions that complete a task, always use an emphasized button. In all other cases use a standard button, or a small button (see below).

Rationale: Users need to realize that a certain function completes a task and know about - possibly negative - consequences.

The emphasized button is always the leftmost button if it is a member of a button group.

Usage - Standard Button vs. Small Button

- Use standard size buttons for frequently-used functions.
- Use small size buttons for seldom-used functions.
- Use small buttons in (exceptional) cases where space is scarce.
- Do not mix both sizes within groups of elements.

Design-relevant Attributes

All buttons are available in an enabled and disabled state (Boolean attribute disabled).



Usage - Disabled Buttons vs. Invisible Buttons

Disabled buttons indicate that a function is not available. Therefore, use disabled buttons for functions that are temporarily disabled. For example, a certain system state, such as an error, may prevent a user from executing a function.

Invisible buttons are buttons that are never available for the user, for example because he or she does not have the permission to perform a certain action.

Positioning Buttons

Buttons are used for explicit functions that refer to a given object or serve for navigational purposes. See Figure 1.

Positioning

Place buttons below the object they refer to. If space is scarce, place the buttons to the right of the object (for several objects place them to the right of the bottom object).

How to make clear which object(s) a button refers to:

- Place the buttons inside, or close to the object.
Example: Place buttons inside group boxes, place buttons close to the fields they refer to.

Uniform Resource Name (URN)

History

The History folder contains links to pages you've visited, for quick access to recently viewed pages.


Days to keep pages in history  [Clear History](#)

Figure 3: Example of a button inside a group referring to a list box

- Left-align button and object.

Example: Left align buttons referring to a field with the field label:

Temporary internet files

Pages you view on the internet are stored in a special folder for quick viewing later

[Delete Files...](#) [Settings...](#)

Figure 4: Example of left-aligned buttons inside a group


- Place button(s) on the same level (area, group box) as the objects which are affected by the action.

Example: Place buttons that refer to fields in a group box or area within that group box, or area (figure 3 and 4).

- Show/hide objects: When an object is hidden, buttons are also hidden.

Example: If a table is hidden, the related buttons are also hidden.

The History folder contains links to pages you've visited, for quick access to recently viewed pages.

Days to keep pages in history  [Clear History](#)

[Save all changes](#) [Cancel](#)

Figure 5: Example of a left-aligned button group containing an emphasized button

- If an emphasized button (see Types) is a member of a button group, it is the leftmost button in this group.
- Navigational buttons are placed at the bottom left of a screen (or screen area).

Overview of Positioning Rules

The following table summarizes the rules for button placement.

Object	Example	Placement
Single Object	Field	Right to the object

Uniform Resource Name (URN)


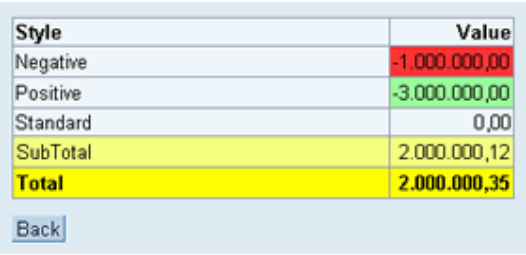
		 <p>Figure 6: Button next to field</p>
Several Objects	Field Group	<p>Default case: left-aligned below the bottom object</p> <p>If space is scarce: to the right of the bottom object (see figure 4).</p>
Area, tabstrip, group box	Group Box	At the bottom, left-aligned (see figure 3)
Table View (fixed size)	Table based on Table View control or Portal Data Viewer	<p>Below the table, left-aligned with the table.</p>  <p>Figure 7: Button below a table</p>
Special case: Long table	Scrolling table	<p>Above and below the table, left-aligned</p> <p>Alternative: Implement a special frame for buttons above a table, which does not scroll.</p>

Table 1: Rules for button placement

Usage guidelines for the different button types and sizes are presented in Button.

Further Information

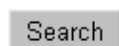
[Control API \[Page 33\]](#), [Browser Support & Accessibility \(508\) \[Page 33\]](#)

Related Controls

[Link \[Page 33\]](#), [Input Field \[Page 33\]](#), [Group \[Page 33\]](#), [Table View \[Page 33\]](#)

3.4.2.6.2.2 Browser Support & 508

In Netscape 4.X buttons will be displayed as standard HTML buttons.

**Figure 1:** Netscape button

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the button control in Internet Explorer 5 and above (Netscape 4.7 uses standard HTML buttons):

Group	Style	IE5 and above
Button Styles	Text Padding	X
	Text Decoration	X
	Font Weight	X
	Border Width and Style	X
Standard Buttons	Standard Background Color	X
	Standard Border Color	X
	Standard Font Color	X
	Standard Hover Color	X
	Disabled Standard Background Color	X
	Font Color of Disabled Standard	X
Emphasized Buttons	Emphasized Background Color	X
	Emphasized Border Color	X
	Emphasized Font Color	X
	Emphasized Hover Color	X
	Disabled Emphasized Background Color	X
	Disabled Emphasized Font Color	X
Standard-sized Buttons	Height	X
Small-sized Buttons	Small Height	X
Text Wrap	White Space	X

Table 1: Editable styles for the button control

For common styles see section HTMLB Controls and Style Editor in Customer Branding and Style Editor.

Accessibility & 508 Support

- Keyboard
The button inserted into the accessibility hierarchy by default - including the button state (for example, disabled) and type (for example, emphasized).
- Default Description
Is provided by the HTMLB rendering engine.
- Application-specific Description
Set an additional description using the `setTooltip` method if needed.

Uniform Resource Name (URN)

An additional description is needed if users need more specific information or instructions. In general, the description has to be extended if a button introduces an interaction that cannot be recognized by a blind user. For example, the descriptions needs to be extended if the button opens a new window.

3.4.2.6.3 Button Row

Definition

The button row control is a container, that displays several buttons in horizontal order. The advantage of the button row control over multiple usage of the "button" control (which would place the buttons also next to each other) is, that the button row control places the buttons closer to each other and that the distance is constant, regardless of the style sheet settings.

When the button row control is not used in a gridlayout or formlayout, it is always rendered in a new line.

- **addButton**
Adds a button to the button row control.
- **id**
Identification name of the button row.

Attributes	M	Values	Usage
addButton		Component	<i>Taglib</i> no tag available <i>Classlib</i> addButton(Button button)
id	*	String (cs)	<i>Taglib</i> id="OrderConfirm" <i>Classlib</i> setId("OrderConfirm")

Example

using the taglib

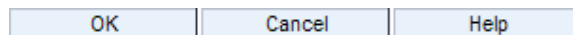
Uniform Resource Name (URN)

```
<hbj:buttonRow
  <hbj:button id="OKButton"
    text="OK"
    width="125px"
  />
  <hbj:button id="CancelButton"
    text="Cancel"
    width="125px"
  />
  <hbj:button id="HelpButton"
    text="Help"
    width="125px"
  />
</hbj:buttonRow>
```

using the classlib

```
Form form = (Form) this.getForm();
ButtonRow br = new ButtonRow();
Button OKButton = new Button("OKButton", "OK");
OKButton.setWidth("125px");
OKButton.setOnClick("ProcessOK");
Button CancelButton = new Button("CancelButton", "Cancel");
CancelButton.setWidth("125px");
CancelButton.setOnClick("ProcessCancel");
Button HelpButton = new Button("HelpButton", "Help");
HelpButton.setWidth("125px");
HelpButton.setOnClick("ProcessHelp");

br.addButton(OKButton);
br.addButton(CancelButton);
br.addButton(HelpButton);
form.addComponent(br);
```

Result**3.4.2.6.4 Chart****Definition**

A control to visualize data in annotated diagrams.

- **axisMaxVal**

Used to calculate the annotation and scaling of the chart. 'axisMaxVal' specifies the maximum value the axis is annotated with. If 'axisMaxVal' is not specified or a value is specified that is less than the maximum value provided by the model, 'axisMaxVal' is set to the maximum value of the model.

- **axisMinVal**

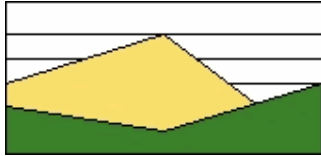
Uniform Resource Name (URN)

Used to calculate the annotation and scaling of the chart. 'axisMinVal' specifies the minimum value of the axis. If 'axisMinVal' is not specified or a value is specified that is greater than the minimum value provided by the model, 'axisMinVal' is set to 0.

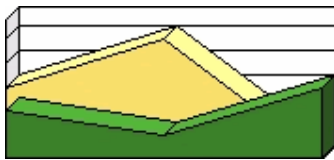
- **chartType**

Controls the style in which the data is displayed.

- AREA



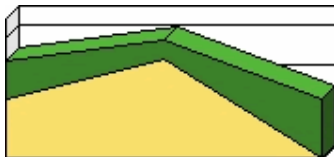
- AREA3D



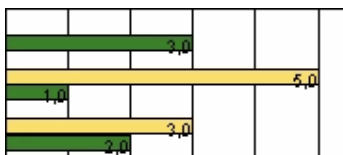
- AREA_STACKED



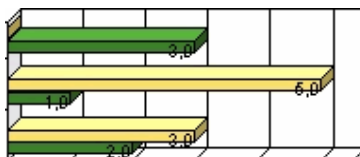
- AREA_STACKED_3D



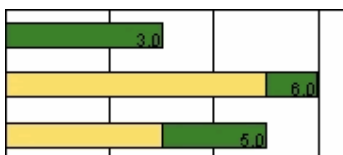
- BARS



- BARS_3D

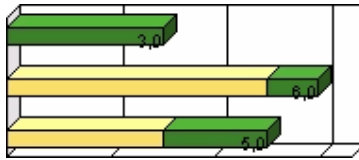


- BARS_STACKED

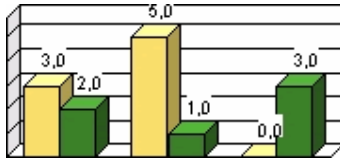


Uniform Resource Name (URN)

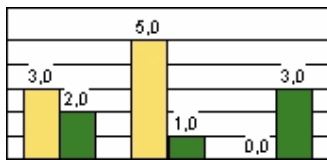
- BARS_STACKED_3D



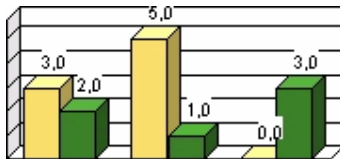
- BITMAP



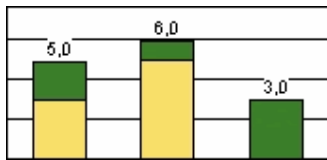
- COLUMNS



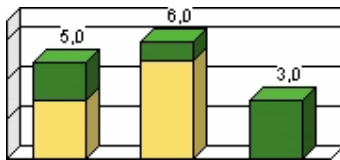
- COLUMNS_3D



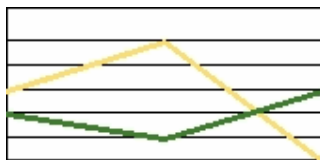
- COLUMNS_STACKED



- COLUMNS_STACKED_3D

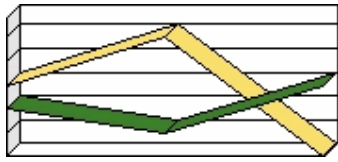


- LINES



- LINES_3D

Uniform Resource Name (URN)



- PIE



- PIE_3D



- PIE_EX



- PIE_EX_3D



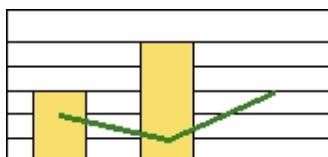
- PIE_SPLIT



- PYRAMID



- TREND



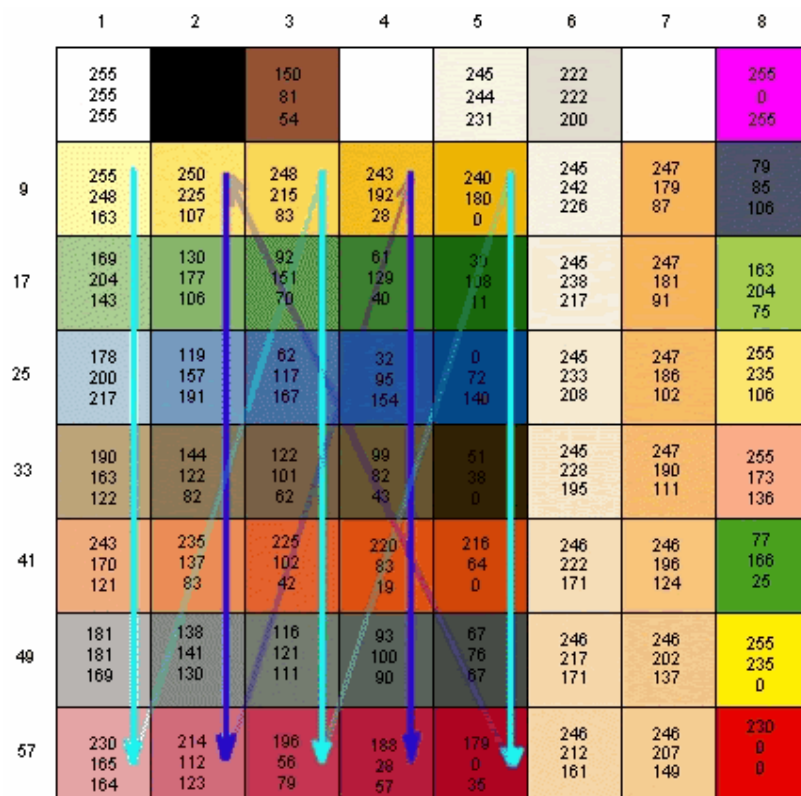
Uniform Resource Name (URN)

- **colorOrder**

The various types of the chart control all use the same set of colors to visualize the values of a data set, but explore the space of possible colors on different paths. The following pictures show the three predefined color schemes and the chart types using them.

- STRAIGHT

This color scheme is used by the various area, column and bar chart graphs.



- SNAKE

This color scheme is used by the pie chart graphs.

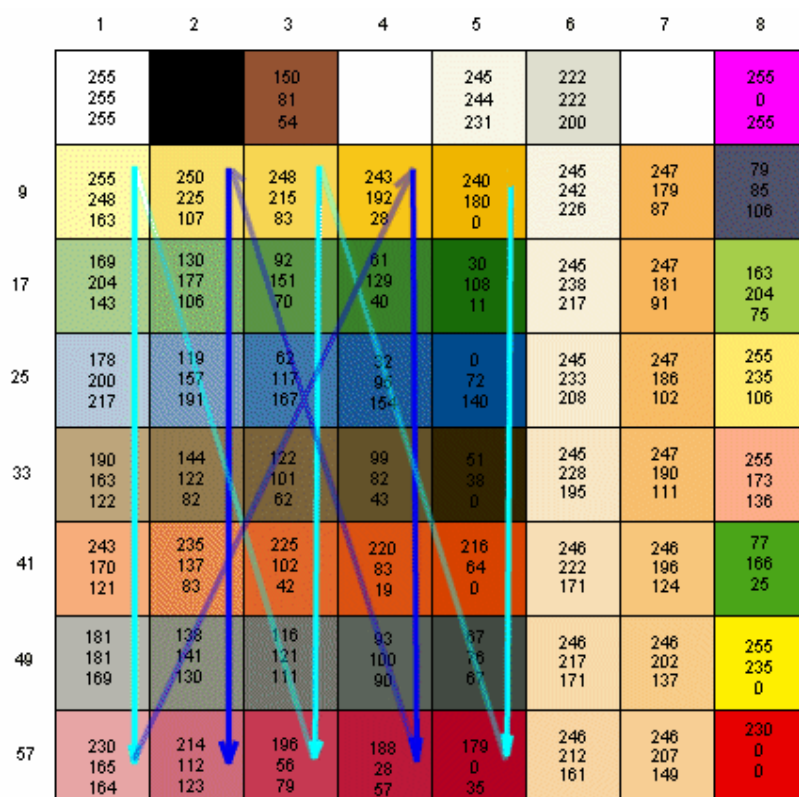
Uniform Resource Name (URN)

	1	2	3	4	5	6	7	8
	255 255 255		150 81 54		245 244 231	222 222 200		255 0 255
9	255 248 163	250 225 107	248 215 83	243 192 28	246 186 0	245 242 226	247 179 87	79 85 106
17	169 204 143	130 177 106	92 151 70	61 129 40	30 108 11	245 238 217	247 181 91	163 204 75
25	178 200 217	119 157 191	62 117 167	32 95 154	0 72 140	245 233 208	247 186 102	255 235 106
33	190 163 122	144 122 82	122 101 62	99 82 43	51 38 0	245 228 195	247 190 111	255 173 136
41	243 170 121	235 137 83	225 102 42	220 83 19	216 64 0	246 222 171	246 196 124	77 166 25
49	181 181 169	138 141 130	116 121 111	93 100 90	67 76 67	246 217 171	246 202 137	255 235 0
57	230 165 164	214 112 123	196 56 79	188 28 57	179 0 35	246 212 161	246 207 149	230 0 0

- REVERSE

This color scheme is used by the line chart graphs.

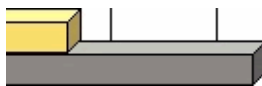
Uniform Resource Name (URN)



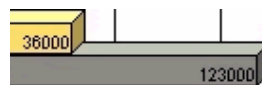
- displayObjectValues**

A boolean value that controls if the values is displayed with the object.

displayObjectValues="false"



displayObjectValues="true"



Not all 'ChartType' settings support the display of values. The example pictures in the 'chartType' attribute description show which types support the display of values.

- height**

Defines the overall height of the chart. The height includes the 'title', 'titleValues' and 'legendPosition'.

- id**

Identification name of the chart.

- legendPosition**

Controls the position of the legend.

- EAST

Places the legend on the right side of the chart.

- NONE

Uniform Resource Name (URN)

- The legend will be suppressed.
 - NORTH
 - Places the legend on top of the chart.
 - SOUTH
 - Places the legend under the chart.
 - WEST
 - Places the legend left of the chart.
- **model**
 - Defines the model which provides the chart with data. How to work with the IChartModel.
- **title**
 - Specifies the headline of the chart.
- **titleCategories**
 - Specifies the axis title for the categories.
- **titleValues**
 - Specifies the axis title for the values.
- **visible**
 - A boolean value that defines if the chart is visible.
- **voidValue**
 - Defines a value that will not be drawn in the chart. Is a "voidValue" set, for example to 0, a line chart would stop on the previous value and start a new line with the next value.
- **voidValueSet**
 - Switches the "voidValue" function on and off.
- **width**
 - Defines the width of the chart. The width include 'titleCategories' and the 'legendPosition'.

Attributes	M	Values	Usage
axisMaxVal		Numeric	<i>Taglib</i> axisMaxVal="2000" <i>Classlib</i> setAxisMaxVal (2000)
axisMinVal		Numeric	<i>Taglib</i> axisMinVal="100" <i>Classlib</i> setAxisMinVal (100)
chartType		AREA AREA_3D AREA_STACKED AREA_STACKED_3D BARS BARS_3D (d)	<i>Taglib</i> chartType="PIE" <i>Classlib</i> setChartType (ChartType.PIE)

Uniform Resource Name (URN)

		BARS_STACKED BARS_STACKED_3D BITMAP COLUMNS COLUMNS_3D COLUMNS_STACKED COLUMNS_STACKED_3D LINES LINES_3D PIE PIE_3D PIE_EX PIE_EX_3D PIE_SPLIT PYRAMID TREND	
colorOrder		DEFAULT (d) STRAIGHT REVERSE SNAKE	<i>Taglib</i> colorOrder="SNAKE" <i>Classlib</i> setColorOrder (ChartColorOrder.SNAKE)
displayObjectValues		FALSE (d) TRUE	<i>Taglib</i> displayObjectValues="TRUE" <i>Classlib</i> setDisplayObjectValues(true)
height		Unit (200)	<i>Taglib</i> height="300" <i>Classlib</i> setHeight("300")
id	*	String (cs)	<i>Taglib</i> id="VacationPlanner" <i>Classlib</i> setId("VacationPlanner")
legendPosition		EAST NONE NORTH SOUTH WEST	<i>Taglib</i> legendPosition="SOUTH" <i>Classlib</i> setLegendPosition (ChartLegendPosition.SOUTH)
model		Component	<i>Taglib</i> model=" myBean.model [Page 33] " <i>Classlib</i> setModel((IChartModel [Page 33]) model)
title		String	<i>Taglib</i> title="Bill board chart" <i>Classlib</i> setTitle("Bill board chart")
titleCategories		String	<i>Taglib</i>

Uniform Resource Name (URN)

			titleCategories="Brand" <i>Classlib</i> setTitleCategories("Brand")
titleValues		String	<i>Taglib</i> titleValues("Overview") <i>Classlib</i> setTitleValues("Overview")
visible		FALSE TRUE (d)	<i>Taglib</i> visible="FALSE" <i>Classlib</i> setVisible(false)
voidValue		Numeric – Double	<i>Taglib</i> no tag available <i>Classlib</i> setVoidValue(10.5)
voidValueSet		FALSE TRUE (d)	<i>Taglib</i> no tag available <i>Classlib</i> setVoidValueSet(false)
width		Unit (500)	<i>Taglib</i> width="400" <i>Classlib</i> setWidth("400")

Example

using the taglib

```

<hbj:chart
  id="myChart1"
  model="myChartBean.model"
  visible="true"
  displayObjectValues="true"
  titleCategories="Company"
  titleValues="Turnover"
  title="Washers by Companies!"
  chartType="BARS_3D"
  legendPosition="EAST"
  colorOrder="STRAIGHT"
/>

```

using the classlib. For information about setting up the bean, see "[IChartModel](#) [Page 33]".

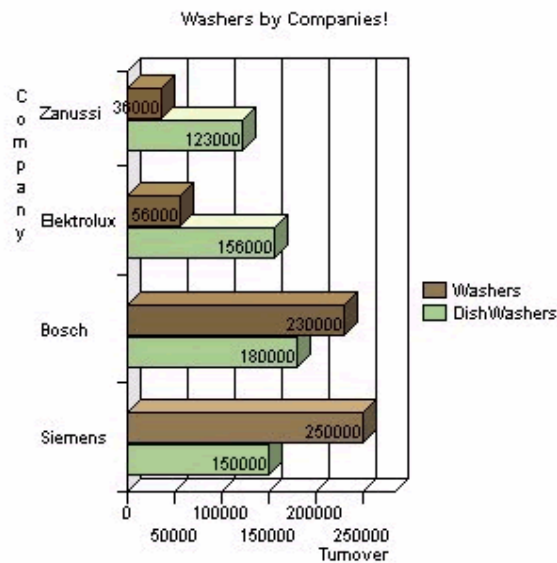
Uniform Resource Name (URN)

```

Form form = (Form) this.getForm();
Chart myChart = new Chart();
myChart.setVisible(true);
myChart.setDisplayObjectValues(true);
myChart.setTitleCategories("Company");
myChart.setTitleValues("Turnover");
myChart.setTitle("Washers by Companies!");
myChart.setChartType(ChartType.BARS_3D);
myChart.setLegendPosition(ChartLegendPosition.EAST);
myChart.setColorOrder(ChartColorOrder.STRAIGHT);

MyVecBean myVecBean = new MyVecBean();
IChartModel chartModel = myVecBean.getModel();
myChart.setModel(chartModel);
form.addComponent(myChart);

```

Result**3.4.2.6.4.1 Usage & Type**

The chart control displays a chart; it offers a variety of different chart types.

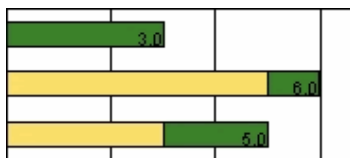


Figure 1: A stacked bar chart as an example of a chart control

Usage

A chart displays data that are relevant for the user in a graphical representation so that the characteristics of the data and their relations are easy to capture for the user.

In cases where it is important for users to know the exact values behind the data, an alternate view may present the data in a table as numbers or texts. A button should allow users to toggle between the diagram and the table view.

Note: For thorough information on charts and their uses see Recommendations for Charts and Graphics in the SAP Design Guild.

Positioning

A chart can be presented in an iView; in this case, it should comprise the main part of the iView. A chart can be combined with other screen elements to allow for interaction with the chart.

Legend

A legend explains the colors used in a chart. For the chart control the legend is generated automatically. It can be placed to the right (preferable) or below the graph; other positions are also possible, but should not be used. The position of the legend is set by the attribute `legendPosition` using the values EAST, NORTH, SOUTH, WEST, or NONE for no legend.

Order of Screen Elements

If there are further interaction elements, obey the following order:

- A filter or shuffler can be placed above the chart, if data can be selected from several sets or if the amount of data has to be reduced.
- Then follows the chart.
- Place legends or other text right to the image or below it, depending on the format of the chart (see above)
- Place pushbuttons for chart-related functionality and status information (for example, zoom factor) below the chart and left align them.
- If there exists an alternative table view, a button below the table allows to toggle between diagram and table view.

Functionality

Typical functionality, which charts may offer, are:

- Switch between chart view and table view
- Zooming and panning (not available for the chart control)
- Drill-down

Using Chart Types - Overview

The chart types are shown in the [Control API description \[Page 33\]](#). The following table overview presents usage hints for the available chart types.

Chart Type	Typical Applications	Variants, Remarks
Area	Cumulated totals (numbers or percentages) over time.	Percentage, Cumulative
Column/Bar	Observations over time or under different conditions; data sets must be small.	Vertical (columns), horizontal (bars); multiple columns/bars, columns/bars centred at zero
Segmented Column/Bar	Proportional relationships over time.	May be scaled to 100%.
Line, Curve	Trends, functional relations.	Data point connected by lines or higher order curves.
Pie	Proportional relationships at a point in time.	Segments may be pulled out of the pie for emphasis (exploded pie chart).

Table 1: Chart types and their applications and variants

Note that there are chart types that may better fit the intended purpose than the available ones. For more information, consult Recommendations for Charts and Graphics in the SAP Design Guild.

Design-relevant Attributes

Look and behavior of the chart control can be controlled by a number of attributes:

- Position and Visibility of Legend: Attribute `legendPosition` allows to hide or show and position the legend (values NONE, EAST, NORTH, SOUTH, WEST).
- Color Order: Use attribute `colorOrder` to control the sequence of colors (values are DEFAULT, STRAIGHT, REVERSE, and SNAKE).
- Height and Width: Attributes `height` and `width` allow to set the size of the chart.
- Display of Values and Titles: A number of attributes is at your disposal to control the look and position of labels for values and categories. For more information see page [Control API for Chart](#).

For detailed information on attributes see the [Control API description \[Page 33\]](#).

Related Topics

[Image \[Page 33\]](#)

[Table View \[Page 33\]](#)

3.4.2.6.4.2 Browser Support & 508

No known compatibility issues - charts are like images and are displayed correctly with any browser.

Editability in Style Editor

Customers cannot customize charts via the Style Editor. The tool offers no editable styles related to charts placed as portal content.

Accessibility – 508 Support

Charts are like images; therefore, the same measures apply.

- **Keyboard**
Charts are not inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed. Do not use the `setAlt` method that sets the alternate text (alt attribute).

3.4.2.6.5 Checkbox

Definition

A control, consisting of a graphic and associated text, that a user clicks to select or deselect an option. A check mark in the checkBox graphic indicates that the option is selected.

The checkbox control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **checked**
A boolean value that indicates if a checkBox is selected. "True" shows a check mark in a checkBox and indicates that the checkBox is selected, "false" leaves the checkBox empty and indicates that the checkBox is not selected.
- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).
A boolean value that defines if the checkBox is click able. If the checkBox is disabled (enabled = false) it is not selectable. A disabled checkBox has a different background color for the checkBox graphic and if the checkBox is checked the a different color for the check mark.
- **encode**
A boolean value that defines how the checkBox text is interpreted. HTML text formatting commands (for example, `<h1>`, `<i>` etc.) can be used to change the display

Uniform Resource Name (URN)

of the checkBox text. If there are no formatting commands in the checkBox text string, the encode attribute has no effect.

Example:

```
text = "<h1><i>Important</i></h1>"
```

encode = "false" Browser output: ☐ ***Important***

the text string is rendered by interpreting the formatting commands.

encode = "true" Browser output: ☐ `<h1><i>Important</i></h1>`

the formatting commands are displayed and not interpreted.

- **id**

Identification name of the checkBox.

- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the checkbox component.

- **labeled**

Notify the component that a label has assigned to it. See the HTMLB JavaDoc for more details on the LabeledComponent class.

- **onClick**

Defines the event handling method that will be processed when the user clicks on the enabled checkBox. If you do not define a 'onClick' Event the checkBox can be clicked but no event is generated.

- **onClientClick**

Defines the JavaScript fragment that is executed when the user clicks on the checkbox. If both events ('onClick' and 'onClientClick') are specified, the 'onClientClick' event handling method is activated first. By default the 'onClick' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onClick' event handling method with the command:

```
htmlbevent.cancelSubmit=true;
```

The 'onClientClick' event is useful to pre process the form and only send the form to client if the preprocessing was successful (for example, date validation, valid number format etc.) to save client/server interaction.

- **text**

Defines the string of text placed right of the check box graphic. If no text should be displayed an empty string (null) can be used. See 'encode' for a formatting example with embedded HTML commands.

- **tooltip**

Defines the hint of the checkBox which is displayed as the mouse cursor passes over the checkBox, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
checked		TRUE	<i>Taglib</i>

Uniform Resource Name (URN)

		FALSE (d)	<code>checked = "TRUE"</code> <i>Classlib</i> <code>setChecked (true)</code>
enabled*		TRUE (d) FALSE	<i>Taglib</i> <code>disabled = "TRUE"</code> <i>Classlib</i> <code>setEnabled (false)</code>
encode		TRUE (d) FALSE	<i>Taglib</i> <code>encode = "FALSE"</code> <i>Classlib</i> <code>setEncode (false)</code>
id	*	String (cs)	<i>Taglib</i> <code>id = "CheckCPU"</code> <i>Classlib</i> <code>setId ("CheckCPU")</code>
jsObjectNeeded**		TRUE FALSE (d)	<i>Taglib</i> <code>jsObjectNeeded = "TRUE"</code> <i>Classlib</i> <code>setJsObjectNeeded (true)</code>
labeled		TRUE FALSE (d)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setLabelled (true)</code>
text		String	<i>Taglib</i> <code>text = "CPU status"</code> <i>Classlib</i> <code>setText ("CPU status")</code>
tooltip		String	<i>Taglib</i> <code>tooltip = "Check CPU status"</code> <i>Classlib</i> <code>setTooltip ("Check CPU status")</code>

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the Component [component \[Page 33\]](#).

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Events	M	Values	Usage
onClick		String (cs)	<i>Taglib</i> <code>onClick = "process_checkbox"</code> <i>Classlib</i> <code>setOnClick ("process_checkbox")</code>

Uniform Resource Name (URN)

onClientClick		String (cs)	<i>Taglib</i> onClientClick="alert('Click') " <i>Classlib</i> setOnClientClick("alert('Click') ")
---------------	--	-------------	--

Example

using the taglib

```
<hbj:checkbox
  id="CheckCPU"
  text="CPU status"
  tooltip="Check CPU status"
  disabled="false"
  checked="true"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
Checkbox cb = new Checkbox("myCheckbox");
cb.setText("CPU status");
cb.setTooltip("Check CPU status");
cb.setDisabled(false);
cb.setChecked(true);
form.addComponent(cb);
```

Result

☒ CPU status

3.4.2.6.5.1 Usage & Type

Checkboxes offer one or multiple choices to the user. The user can select none, one, or as many options as desired in a group of checkboxes.

Include the following search sources

☒ SAP R/3
☐ Intranet
☒ Lotus Notes
☐ Google
☒ Yahoo
☐ HotBot

Figure 1: A checkbox group

Usage

Checkboxes are the appropriate elements when users can choose between multiple options. They can appear as a single checkbox or grouped.

In a checkbox group the choices are not exclusive, that is, a user can check several options in a group. If you need single-selection use radio buttons or a dropdown list box, instead.

Checkbox Group

For groups of checkboxes use the checkbox group control if applicable. This control allows to arrange checkboxes in one column, one row, or in a matrix-like fashion.

Note: It is not possible to determine the horizontal spacing within a checkbox group. If you need a different spacing than that supplied by the checkbox group control, use single checkboxes and a grid layout control if applicable.

Arrangement and Design Alternatives

Checkboxes offer one or multiple choices to the user. The user can select none, one, or as many options as desired in a group of checkboxes.

Include the following search sources

☒ SAP R/3 ☐ Intranet ☒ Lotus Notes
☐ Google ☒ Yahoo ☐ HotBot

Checkbox groups offer users a set of multiple options that may be arranged either horizontally (2-3 checkboxes), vertically (not more than about 12 checkboxes), or in a matrix-like fashion. Note that checkbox groups are appropriate for static and relative small numbers of options only. Use the table view for larger and dynamic option sets.

For the alignment of checkboxes we distinguish the following cases:

- Checkboxes that refer to adjacent fields.
- Checkboxes that do not refer to elements but should be included in field groups.
- Checkboxes that can be arranged as an independent block of information

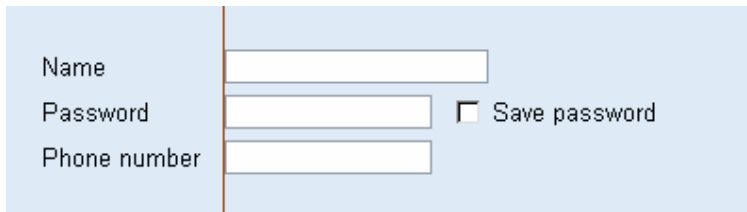
Case 1: Checkboxes that Refer to One or More Fields

The figure consists of two side-by-side form examples. The left example shows a form with three input fields: 'Name', 'Password', and 'Phone number'. A checkbox labeled 'Save password' is positioned to the right of the 'Password' field. A red arrow points to this checkbox with the text 'Alignment of Checkboxes'. The right example shows a form with five input fields: 'First name', 'Last name', 'Gender' (a dropdown menu showing 'male'), 'City', and 'Street'. A checkbox labeled 'Same address as last order' is positioned to the right of the 'City' and 'Street' fields. A red arrow points to this checkbox with the text 'Alignment of Checkboxes'.

Figure 1a-b: Checkbox that refers to one input field above it (left) or to two (City and Street, right)

Alternatively, a single checkbox can be placed to the right of a reference field (figure 1c) if space permits. If there is more than one reference field place the checkbox right to the bottom reference field.

Uniform Resource Name (URN)



Name

Password ☐ Save password

Phone number

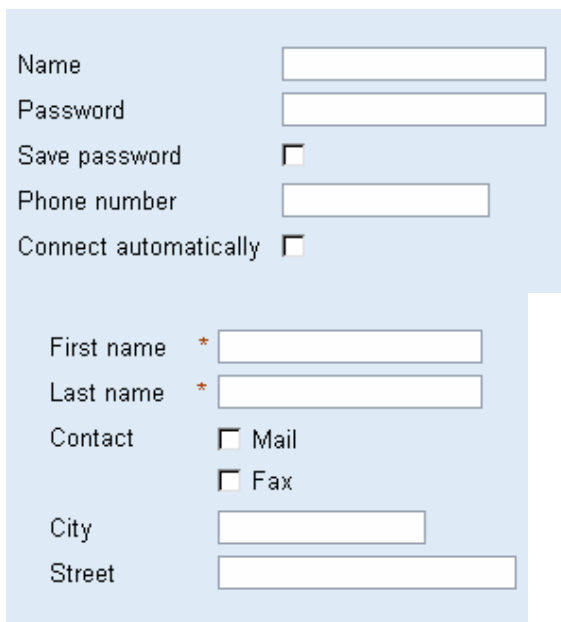
Figure 1c: Checkbox that refers to an input field left of it (equivalent to figure 1a)

Case 2: Checkboxes that are Included in a Field Group

If checkboxes are included in a field group but do not refer to a certain field, place the checkbox labels to the left and align the checkboxes themselves with the other input fields (figure 2a).

Note: In this case you have to set the checkbox text to an empty text and use the label control for the label.

Alternatively, you can add a label that is left-aligned with the other labels of the group and use the checkbox text for additional information (figure 2b). In that case, only the first checkbox should have a label that describes the whole group.



Name

Password

Save password ☐

Phone number

Connect automatically ☐

First name *

Last name *

Contact ☐ Mail

☐ Fax

City

Street

Figure 2a-b: Checkbox within a field group, either with label to the left (left), or with two labels

If space permits you can alternatively use a horizontal checkbox group that occupies one line (typically for 2-3 checkboxes, figure 2c).

Uniform Resource Name (URN)

Figure 2c-d: Horizontal checkbox group within a field group (left); indented checkbox without group label (right). Figure 2d shows wrong alignment because the checkbox does not refer to the password field.

Note: Do not use an arrangement without a group label in this case (figure 2d) because it may lead to misinterpretations. Such a layout suggests a dependency from the field above the group to the user. Even though the layout in figure 2d is the same as in 1a, the usage is incorrect because the checkbox does not refer to the password field. Therefore, use it only if such a dependency does exist (case 1).

Case 3: Checkboxes that Form an Independent Information Block

If checkboxes are arranged in a checkbox group, they are left aligned with other labels and arranged in a matrix-like fashion. Such groups have either to be included in a group control (see figure 3a) or separated from the field group by white space (figure 3b).

Figure 3a-b: Checkbox group that forms an information unit of its own - either included in a group (left) or separated by an empty line (right)

Instead of a matrix, you can use a horizontal arrangement if there are only few checkboxes. In this case, set the columnCount attribute of the checkbox group control to a value that results in one row only.

There are two possible arrangements for horizontal checkbox groups:

- The checkbox row can be introduced by a label to the left - in this case align the checkboxes with other elements and use the label control for the label (figure 4a)
- The checkbox row does not have an introductory label (figure 4b)

Uniform Resource Name (URN)

Separate the horizontal checkbox group from preceding fields by an empty line.

Note: An "extreme" case of a horizontal checkbox group is a single checkbox.

First name *

Last name *

City

Street

Contact ☐ Mail ☐ Fax

Name

Password

Phone number

☐ Save password ☐ Connect automatically

Figure 4a-b: Example of a horizontal checkbox group, either with an introductory label to the left (left), or without an introductory label (right).

For more than two to three checkboxes a vertical arrangement with or without label may be more appropriate (see figure 5a and 5b).

Uniform Resource Name (URN)

First name *

Last name *

City

Street

Contact

- ☐ Mail
- ☐ Phone
- ☐ Fax
- ☐ E-mail

Name

Password

Phone number

☐ Save password

☐ Connect automatically

☐ Disconnect of idle for 10 minutes

Figure 5a-b: Example of a vertical checkbox group, either with an introductory label to the left (left), or without (right).

Dependent Fields

In some cases, the state of input fields, dropdown list boxes, or other controls may depend on the setting of a checkbox. Below we present a simple example where users may enter their contact preferences (figure 6a). An unchecked checkbox describes the default case; it is set the input field below it is read-only. A checked checkbox describes the less frequent case. If the user checks the checkbox, the input elements are ready for input.

First name *

Last name *

Contact

- ☒ Mail
- City/Zip Code
- Street
- ☒ E-mail
-

Figure 6: Checkboxes that controls the editability of the input field(s) below the checkbox

Uniform Resource Name (URN)

If there are more dependent elements indent the dependent group so that their labels are left aligned with other input fields (top checkbox). If there is only one dependent field, usually a field label is not needed (bottom checkbox).

Design-relevant Attributes

Checkboxes have the disabled and checked attributes. Set disabled to TRUE if a checkbox cannot be checked or unchecked by a user temporarily. Set checked to TRUE to preset a checkbox to the checked state. Use attribute text to set the descriptive label text for a checkbox.

You can also set the column count for checkbox groups (attribute columnCount).

Related Controls

[Radio Button \[Page 33\]](#), [Dropdown List Box \[Page 33\]](#), [List Box \[Page 33\]](#), [Label \[Page 33\]](#), [Grid Layout \[Page 33\]](#)

3.4.2.6.5.2 Browser Support & 508

The checkbox renders in every supported browser.

Editability in Style Editor

The checkbox itself renders as the standard browser control. Style Editor changes can be made to the corresponding label.

Checkbox Groups

There is no editability for checkbox groups in the style editor.

Accessibility – 508 Support

Checkboxes have to be used in combination with the label control, which points to the assigned checkbox if they are used with a label to the left of the checkbox. This ensures that screen readers are aware of the relationship between the both elements and can read the correct label to the according checkbox.

- Keyboard
Checkboxes are inserted into the accessibility hierarchy by default.
- Default Description
Is provided by the HTMLB rendering engine.
- Application-specific Description
Set an additional description using the `setTooltip` method if needed.
- Label

Uniform Resource Name (URN)

Has to be connected to a label control for left-hand labels (use method `setLabelFor` for identifying the corresponding checkbox or checkbox group).

3.4.2.6.6 Date Navigator

Definition

A control for advanced handling of all actions which require a date input and to visualize a date.

- **id**
Identification name of the dateNavigator.
- **model**
Defines the model which provides the dateNavigator with data. How to work with the [DateNavigatorModel \[Page 33\]](#).
- **monthPerColumn**
The dateNavigator can display several month. The months are arranged in matrix form. This attribute defines the number of columns of the matrix.
- **monthPerRow**
The dateNavigator can display several month. The months are arranged in matrix form. This attribute defines the number of rows of the matrix.
- **onNavigate**
The navigation fields are located left and right of the displayed month `<< November 2001 >>`. The `<<` and `>>` fields can be used to select the previous and next month. If a dateNavigator has more columns the previous month navigator is located at the first column and the next month navigator at the last column.

The 'onNavigate' attribute defines the event handling method that will be processed when the user clicks on the navigation fields.
- **onDayClick**
Defines the event handling method that will be processed when the user clicks on a day.
- **onWeekClick**
Defines the event handling method that will be processed when the user clicks on a week. The week is the first column of the dateNavigator grid.
- **onMonthClick**
Defines the event handling method that will be processed when the user clicks on the header text string representing the month displayed.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> <code>id = "VacationPlanner"</code> <i>Classlib</i> <code>setId ("VacationPlanner")</code>

Uniform Resource Name (URN)

model [Page 33]	*	String	<i>Taglib</i> model=" myBean.model [Page 33] " <i>Classlib</i> setModel ((DateNavigatorModel [Page 33]) model)
monthPerColumn		Numeric (1)	<i>Taglib</i> monthPerColumn = "3" <i>Classlib</i> setMonthPerColumn (3)
monthPerRow		Numeric (1)	<i>Taglib</i> monthPerRow = "4" <i>Classlib</i> setMonthPerRow (4)

Events	M	Values	Usage
onNavigate		String (cs)	<i>Taglib</i> onNavigate = "ProcessNav" <i>Classlib</i> setOnNavigate ("ProcessNav")
onDayClick		String (cs)	<i>Taglib</i> onDayClick = "DaySel" <i>Classlib</i> setOnDayClick ("DaySel ")
onWeekClick		String (cs)	<i>Taglib</i> onWeekClick = "WeekSel" <i>Classlib</i> setOnWeekClick ("WeekSel ")
onMonthClick		String (cs)	<i>Taglib</i> onMonthClick = "MonSel" <i>Classlib</i> setOnMonthClick ("MonSel")

Example

using the taglib

```

<hbj:dateNavigator
  id="myDateNavigator1"
  model="myBean.model"
  monthsPerColumn="2"
  monthsPerRow="3"
  onNavigate="myOnNavigate"
  onDayClick="myOnDayClick"
  onWeekClick="myOnWeekClick"
  onMonthClick="myOnMonthClick"
/>

```

Uniform Resource Name (URN)

using the classlib

```

IPortalComponentRequest request =
    (IPortalComponentRequest) this.getRequest();
IPortalComponentContext context = request.getComponentContext();
IPortalComponentProfile profile = myContext.getProfile();
IPageContext pagecontext =

    PageContextFactory.createPageContext(request);

Form form = (Form) this.getForm();
DateNavigator dn = new DateNavigator(pagecontext);
dn.setId("myDateNavigator1");
dn.setMonthsPerColumn(2);
dn.setMonthsPerRow(3);
dn.setOnNavigate("myOnNavigate");
dn.setOnDayClick("myOnDayClick");
dn.setOnWeekClick("myOnWeekClick");
dn.setOnMonthClick("myOnMonthClick");
MyBean myBean = new MyBean(pagecontext);
dn.setModel(myBean.getModel());
form.addComponent(dn);

```

Result

«« September 2001							Oktober 2001							November 2001 »»						
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
35	27	28	29	30	31	1	2	40	1	2	3	4	5	6	7	44	29	30	31	1
36	3	4	5	6	7	8	9	41	8	9	10	11	12	13	14	45	5	6	7	8
37	10	11	12	13	14	15	16	42	15	16	17	18	19	20	21	46	12	13	14	15
38	17	18	19	20	21	22	23	43	22	23	24	25	26	27	28	47	19	20	21	22
39	24	25	26	27	28	29	30	44	29	30	31	1	2	3	4	48	26	27	28	29
																	30	1	2	
Dezember 2001							Januar 2002							Februar 2002						
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
48	26	27	28	29	30	1	2	1	31	1	2	3	4	5	6	5	28	29	30	31
49	3	4	5	6	7	8	9	2	7	8	9	10	11	12	13	6	4	5	6	7
50	10	11	12	13	14	15	16	3	14	15	16	17	18	19	20	7	11	12	13	14
51	17	18	19	20	21	22	23	4	21	22	23	24	25	26	27	8	18	19	20	21
52	24	25	26	27	28	29	30	5	28	29	30	31	1	2	3	9	25	26	27	28
1	31	1	2	3	4	5	6										1	2	3	

3.4.2.6.1 Usage & Type

«« January 2002 »»						
Su	Mo	Tu	We	Th	Fr	Sa
1	30	31	1	2	3	4
2	6	7	8	9	10	11
3	13	14	15	16	17	18
4	20	21	22	23	24	25
5	27	28	29	30	31	1

Figure 1: Example of the date navigator displaying one month

Uniform Resource Name (URN)

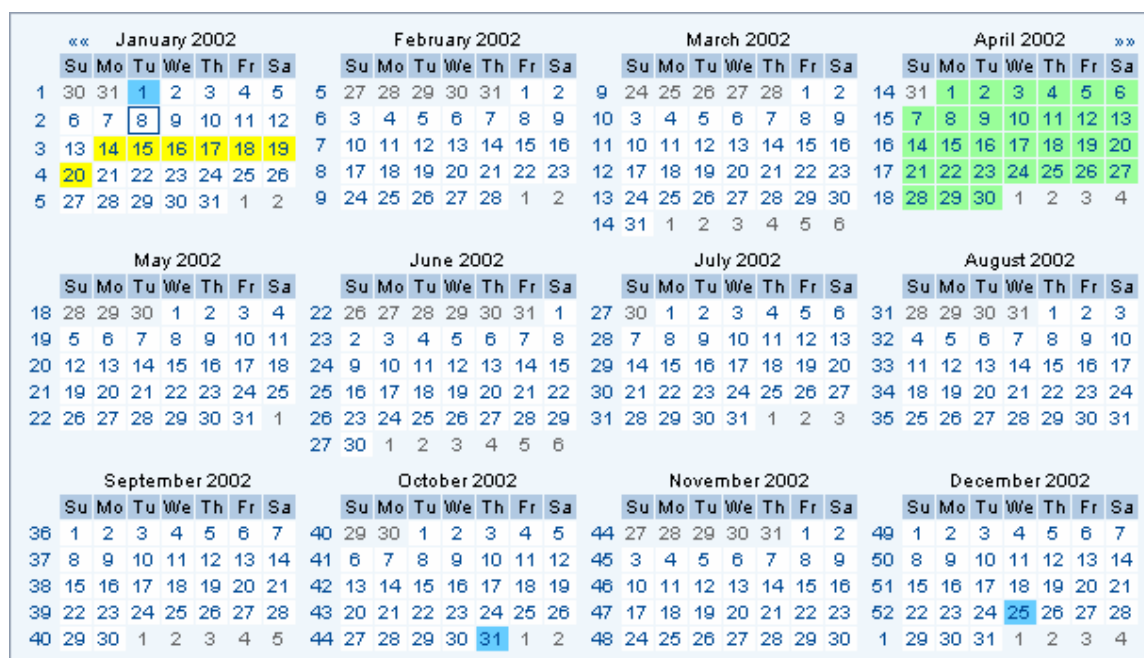


Figure 2: Example of the date navigator displaying 12 months

Usage

The date navigator is a control for advanced handling of all actions, which require a date input and to visualize a date. Thus, the main purpose of the date navigator control is to aid users in inputting a date. It also ensures that the date is entered in an appropriate format. In such cases, it is highly recommended that users also be allowed the option to manually input the date as well.

Note: If the date must be entered in a particular format, an example should be given next to the entry field.

The date navigator can also be used to visualize the Western calendar.

Design-relevant Attributes

The date navigator allows to set the number of months per row (monthsPerRow) and per column (monthsPerColumn).

Related Controls

There are no related controls.

3.4.2.6.6.2 Browser Support & 508

Netscape Navigator 4 cannot display certain visual aspects of the standard date navigator control.

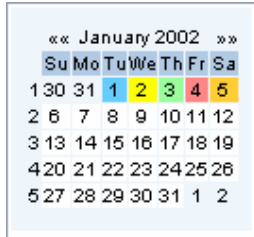


Figure 1: Example of the date navigator in Netscape Navigator 4



Figure 2: Example of the standard date navigator

Editability in Style Editor

In the Style Editor for release 5.0 the date navigator is called "calendar." In the Style Editor, one can change the background colors, text attributes, padding and the type of cursor that appears over clickable elements. Here is a list of the styles that can be changed:

Group	Style	IE5 and above	Netscape 4.7
Day names	Background Color for Days of the Week	x	x
Day Numbers	Entry Width Alignment of Entry Text Decoration of Entry Text Selection Background Color 1 Selection Background Color 2 Selection Background Color 3 Selection Background Color 4 Selection Background Color 5	x x x x x x x x x	 x x x x x
Present Day	Background Color Border	x x	x
Other Days of this Month	Background Color	x	x
Days of Previous or Next Month	Inactive Font Color	x	
Container	Background Color of Container Body Container Border Padding of Container Content	x x x	x

Table 1: Editable styles for the date navigator control

Accessibility - 508 Support

Currently the date navigator has not been adapted to support screen readers. For more information about accessibility, see the SAP Portals Accessibility Guidelines.

3.4.2.6.7 Drag Source

Definition

DragSource is a container that contains controls that can be dragged. This component works together with [dropTarget \[Page 33\]](#) component which defines the container where the dragged item can be dropped off.

- **columnKey**

Defines the column of the dragSource when the dragSource is used in a custom defined cellRenderer of a TableView or inside a Tree. The column key can be recovered with the DropEvent object.

- **dropTargetDesign**

Defines the design of the marker which is used to indicated the dropTarget where the 'flavours' attribut matches. The marker is displayed when a dragSource component has been clicked and dragging has started.

- BORDERED

Displays a frame around the matching dropTarget.

- UNDERLINED

Displays a base line under the matching dropTarget.

- NONE

No indication of the matching dropTarget.

- **flavours**

Flavours are used to identify possible dropTargets. When this 'flavours' attribute matched with the 'flavours' attribute of a dropTarget it is identified as possible drop of target. Drag-and-Drop is only possible if the dragSource contains at least one flavour of the dropTarget. A dragSource can have more than one flavour assigned to it. Use the addFlavour method to add additional flavours.

- **id**

Identification name of the dragSource.

- **scope**

Defines if the drag-and-drop process can take place only inside the current form or in the whole browser.

- Browser

drag-and-drop process can take place in the whole browser.

- Form

drag-and-drop process can take place in the current form.



For the Netscape browser only the scope FORM is supported.

- **sourceContainerName**

Defines the source container of the dragSource when the dragSource is used in a custom defined cellRenderer of a TableView or inside a Tree. The sourceContainerName can be recovered with the DropEvent object.

Uniform Resource Name (URN)

- **value**

Sets the value of the dragSource. The value is transferred from the dragSource to the dropTarget when dropping is complete. The value can be recovered with the DropEvent object.

- **width**

Sets the width of the dragSource container.

Attributes	M	Values	Usage
columnKey		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setColumnKey ("aKey")
dropTargetDesign		BORDERED UNDERLINE NONE	<i>Taglib</i> dropTargetDesign="NONE" <i>Classlib</i> setDropTargetDesign (DropTargetDesign.NONE)
flavours		String (cs)	<i>Taglib</i> flavours="F1" <i>Classlib</i> setFlavours ("F1")
id	*	String (cs)	<i>Taglib</i> id="drag" <i>Classlib</i> setId ("drag")
scope		BROWSER FORM	<i>Taglib</i> scope="BROWSER" <i>Classlib</i> setScope (Scope.Browser)
sourceContainerName		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setSourceContainerName ("s1")
value		String	<i>Taglib</i> value="R2D2" <i>Classlib</i> setValue ("R2D2")
width		Unit	<i>Taglib</i> No tag available <i>Classlib</i> setWidth ("100")

Example

using the taglib

```
<hbj:dragSource
  id="dragid1"
  flavours="F1"
  value="R2D2"
/>
```

Result



3.4.2.6.8 Dropdown List Box

Definition

A control with a dropdown arrow that the user clicks to display a list of options. An item in the dropdownListBox is called listBoxItem. The dropdownListBox supports client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **design**
Sets the design of the dropdownListBox.
- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).
A boolean value that defines if the dropdownListBox is click able. If the dropdownListBox is disabled (enabled = false) it is not selectable. A disabled dropdownListBox has a different color for the displayed listBoxItem.
- **id**

Uniform Resource Name (URN)

Identification name of the dropdownListBox.

- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the dropdownListBox component.

- **labeled**

Notify the component that a label has assigned to it. See the HTMLB Javadoc for more details on the LabeledComponent class.

- **model**

Defines the model which provides the dropdownListBox with data. How to work with the [IListModel \[Page 33\]](#).

- **nameOfKeyColumn**

Specifies the name of the column that contains the key. This is used when you use an underlying table in the model.

- **nameOfValueColumn**

Specifies the name of the column that contains the visible text. This is used when you use an underlying table in the model.

- **onClientSelect**

Defines the JavaScript fragment that is executed when the user clicks on the dropdownListbox. If both events ('onSelect' and 'onClientSelect') are specified, the 'onClientSelect' event handling method is activated first. By default the 'onSelect' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onSelect' event handling method with the command

```
htmlbevent.cancelSubmit=true;
```

The 'onClientSelect' event is useful to pre process the form and only send the form to client if the preprocessing was successful (for example, date validation, valid number format etc.) to save client/server interaction.



A dropdownListbox click usually activates the client/server interaction. If an input field has to be filled out for further processing, the JavaScript fragment can check the necessary input on the client side and display a message if the necessary input is missing, without server interaction.



To use JavaScript the JSP has to use the page tag (see [page \[Page 33\]](#) tag).

- **onSelect**

Defines the event handling method that will be processed when the user clicks on the enabled dropdownListbox. If you do not define a onClick event the dropdownListbox can be clicked but no event is generated.

- **requiresValidation**

A boolean value that defines if the selected value in the dropdownListBox has to be validated before a server event is generated.

- **selection**

Uniform Resource Name (URN)

Specifies the key of the listBoxItem which is displayed in the dropdownListBox.

- **tooltip**

Defines the hint of the dropdownListBox which is displayed as the mouse cursor passes over the dropdownListBox, or as the mouse button is pressed but not released.

- **width**

Defines the width of the dropdownListBox in pixel or percent.

Attributes	M	Values	Usage
design		STANDARD (d) SMALL	<i>Taglib</i> No tag available <i>Classlib</i> setDesign (DropdownListBoxDesign.SMALL)
enabled*		TRUE (d) FALSE	<i>Taglib</i> disabled="TRUE" <i>Classlib</i> setEnabled (false)
id	*	String (cs)	<i>Taglib</i> id = "listbox_te" <i>Classlib</i> setId ("listbox_te")
jsObjectNeeded**		TRUE (d) FALSE	<i>Taglib</i> jsObjectNeeded = "TRUE" <i>Classlib</i> setJsObjectNeeded (true)
labeled		TRUE FALSE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setLabelled (true)
model [Page 33]		String	<i>Taglib</i> model = " mybean.model [Page 33] " <i>Classlib</i> setModel((IListModel [Page 33]) model)
nameOfKeyColumn		String	<i>Taglib</i> nameOfKeyColumn = "k1" <i>Classlib</i> setNameOfKeyColumn ("k1")
nameOfValueColumn		String	<i>Taglib</i> nameOfValueColumn = "v1" <i>Classlib</i> setNameOfValueColumn ("v1")
requiresValidation		TRUE FALSE (d)	<i>Taglib</i> No tag available <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setRequiresValidation(true)</code>
selection		String	<i>Taglib</i> <code>selection = "HD"</code> <i>Classlib</i> <code>setSelection("HD")</code>
tooltip		String	<i>Taglib</i> <code>tooltip = "select an item"</code> <i>Classlib</i> <code>setTooltip ("select an item")</code>
width		Unit	<i>Taglib</i> <code>width = "200"</code> <i>Classlib</i> <code>setWidth ("200")</code>

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the Component [component \[Page 33\]](#).

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Events	M	Values	Usage
onClientSelect		String (cs)	<i>Taglib</i> <code>onClientSelect="alert('Click')"</code> <i>Classlib</i> <code>setOnClientSelect ("alert('Click')")</code>
onSelect [Page 33]		String (cs)	<i>Taglib</i> <code>onSelect="proc_listbox"</code> <i>Classlib</i> <code>setOnSelect ("proc_listbox")</code>

To entries in the dropdownListBox are created with the [listBoxItem \[Page 33\]](#) control.

Example

using the taglib

Uniform Resource Name (URN)

```

<hbj:dropdownListBox
    id="DDCitiesNearby"
    tooltip="Cities surrounding SAP"
    selection="WD"
    nameOfKeyColumn="KeyCol"
    nameOfValueColumn="KeyVal"
    onSelect="ProcessCity"
    onClientSelect="PreprocessCity"
>

    <hbj:listBoxItem key="HD" value="Heidelberg" />
    <hbj:listBoxItem key="HK" value="Hockenheim" />
    <hbj:listBoxItem key="WD" value="Walldorf" />
    <hbj:listBoxItem key="WL" value="Wiesloch" />
</hbj:dropdownListBox>

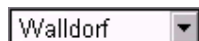
```

using the classlib

```

Form form = (Form) this.getForm();
DropDownListBox ddl = new DropDownListBox("DDCitiesNearby");
ddl.setWidth("300");
ddl.addItem("HD", "Heidelberg");
ddl.addItem("HK", "Hockenheim");
ddl.addItem("WD", "Walldorf");
ddl.addItem("WL", "Wiesloch");
ddl.setOnSelect("ProcessCity");
ddl.setOnClientSelect("PreprocessCity");
ddl.setTooltip("Cities nearby");
ddl.setSelection("WD");
form.addComponent(ddl);

```

Result**3.4.2.6.8.1 Usage & Type**

The dropdown list box is a field with an arrow icon on the right side. Clicking on this icon drops down a list immediately below the field and shows the user which values can be chosen. An entry in the list is called list box item. The dropdown list box is read-only.

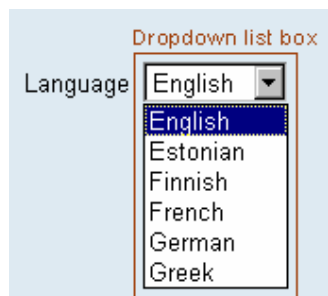


Figure 1: A dropdown list box with six language items

Uniform Resource Name (URN)

See also [List Box – Usage & Type \[Page 33\]](#) for details when to use dropdown list box, list box, item list and table view.

















Usage

Use dropdown list boxes:

- To support the selection of a value from a limited quantity. The number of items should not exceed 20.
- To switch between views on large amounts of data. Especially in iViews, this is a good method to save space. With dropdown list boxes, more views are possible than, for example, in a tabstrip, because the number of views is not limited by space. However, the list should not be longer than about 12 items.
- In a shuffler for filtering a larger data set, in order to get simplified and reduced views of the data. The shuffler mimics natural language statements for formulating the query, but can also be used with query statements consisting of words only. The query statement is typically assembled by combining static texts with dynamic elements like dropdown lists, edit fields and selection elements.

Note: The dropdown list box control does not render a descriptive label automatically. Use the label control to add a description.

The following table shows examples for the usage described above:

<div>Street Address * <input type="text"/></div> <div>City * <input type="text"/></div> <div>Country <input type="text" value="Austria"/></div> <div>Email address * <input type="text"/></div>	Selection of a value																		
<div><div>Contents of documents</div><div>sorted by <div>Name <input type="text"/></div> </div><div><div>documents</div><div>Details New Folder New File New Text File New Link</div><table><thead><tr><th>Name</th><th>Modified</th><th>Size</th></tr></thead><tbody><tr><td> discussions</td><td>1/7/02 4:08:06 PM</td><td>Details Delete</td></tr><tr><td> faqs</td><td>1/7/02 4:07:19 PM</td><td>Details Delete</td></tr><tr><td> Links</td><td>1/7/02 4:04:27 PM</td><td>Details Delete</td></tr><tr><td> news</td><td>1/7/02 4:13:21 PM</td><td>Details Delete</td></tr><tr><td> Public Documents</td><td>1/15/02 10:14:17 AM</td><td>Details Delete</td></tr></tbody></table></div></div>	Name	Modified	Size	 discussions	1/7/02 4:08:06 PM	Details Delete	 faq s	1/7/02 4:07:19 PM	Details Delete	 Links	1/7/02 4:04:27 PM	Details Delete	 news	1/7/02 4:13:21 PM	Details Delete	 Public Documents	1/15/02 10:14:17 AM	Details Delete	View selection
Name	Modified	Size																	
 discussions	1/7/02 4:08:06 PM	Details Delete																	
 faq s	1/7/02 4:07:19 PM	Details Delete																	
 Links	1/7/02 4:04:27 PM	Details Delete																	
 news	1/7/02 4:13:21 PM	Details Delete																	
 Public Documents	1/15/02 10:14:17 AM	Details Delete																	

Uniform Resource Name (URN)

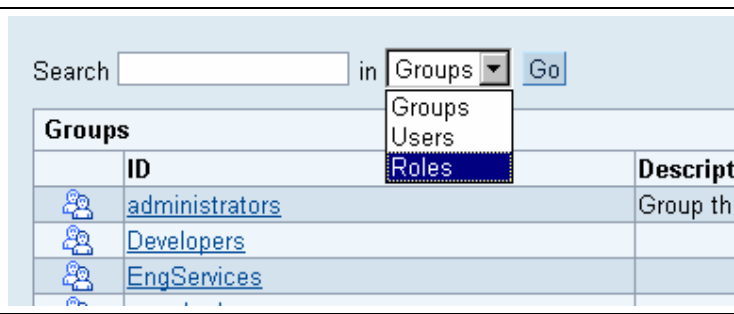
		Dropdown list box used in a shuffler.
--	--	---------------------------------------

Table 1: Usage examples for the dropdown list box**Choosing the Appropriate Selection Control**

A dropdown list box is similar in function to a list box - both offer a list of items where users can select one item from, that is, both are single-selection lists.

See Forms - Using Different List Types for guidelines on choosing the appropriate selection control.

Note: For very small item numbers (2-6) and if the users should immediately see all alternatives, use

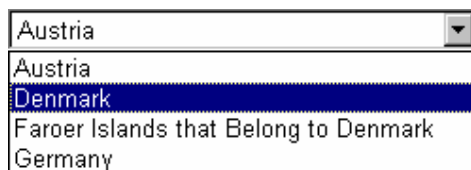
Design-relevant Attributes

The dropdown list box can be set to an enabled or disabled state. Set attribute disabled to FALSE to enable a checkbox, set disabled to TRUE to disable it.

A disabled dropdown list box is not clickable, no item is selectable.

**Figure 2:** Disabled dropdown list box

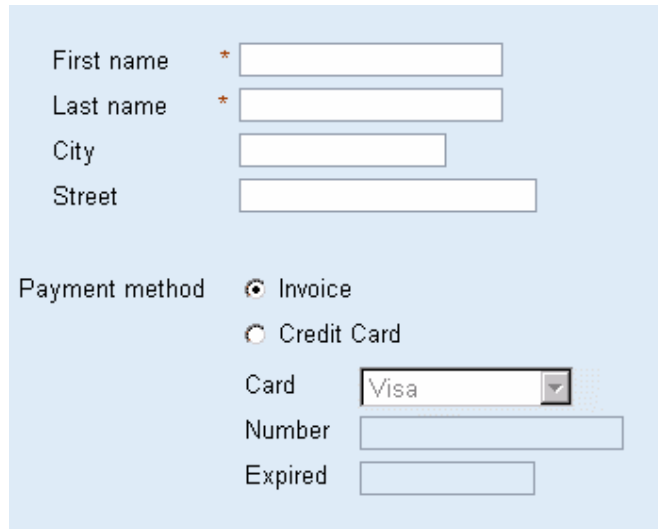
The dropdown list box does not have a width attribute. Note, that this control takes the width from the widest list box item.

**Figure 3:** Dropdown list box with a very wide item**Usage - Disabled State**

Set the disabled state if the user is not allowed to change the value of a dropdown list box or if a larger group of input elements including a dropdown list box is disabled.

Uniform Resource Name (URN)

Example: A set of fields including a dropdown list box is disabled because the user unchecked an option (see figure 4).



First name *

Last name *

City

Street

Payment method ☒ Invoice ☐ Credit Card

Card

Number

Expired

Figure 4: Disabled dropdown list box - the fields are disabled because the user checked the Invoice option

Related Controls

[Input Field \[Page 33\]](#), [Item List \[Page 33\]](#), [Label \[Page 33\]](#), [List Box \[Page 33\]](#), [Radio Button \[Page 33\]](#), [Tree View \[Page 33\]](#)

3.4.2.6.8.2 Browser Support & 508

Netscape 4.7

The disabled version of the dropdown list box is not available for Netscape 4.7.

Netscape 6.1 and 6.2

In Netscape Version 6.1 and 6.2, the dropdown list box looks slightly different than the standard control.

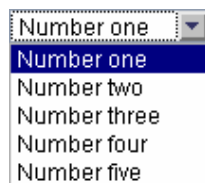


Figure 1: Netscape 6.1/6.2 example of an enabled dropdown list box



Figure 2: Netscape 6.1/6.2 example of a disabled dropdown list box

Editability in Style Editor

In the Style Editor, the dropdown list box does not appear in the list of customizable elements directly. No control-specific styles exist for this element, only common styles are used.

Accessibility - 508 Support

Dropdown list boxes have to be used in combination with the label element which points to the assigned listbox. This ensures, that screen readers are aware of the relationship between the both elements and can read the correct label to the according dropdown list box.

- **Keyboard**
The dropdown list box inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed.
- **Label**
Has to be connected to a label control (use method `setLabelFor` for identifying the corresponding dropdown list box).

3.4.2.6.9 Drop Target

Definition

Defines items on the page where dragged items can be dropped off. This component works together with dropSource component which items can be dragged. Use the `addFlavour` method to add additional flavours.

- **flavours**
Flavours are used to define which dragSources are allowed for this dropTarget. Drag-and-Drop is only possible if the dragSource contains at least one flavour of the dropTarget.
- **id**
Identification name of the dropTarget.
- **onDrop**
Specifies then event handling method that is called when the user drops of the dragged object. If this method is not specified and not defined an exception is raised when the object is dropped.
- **value**
Sets the value of the dropTarget. The value can be recovered with the `DropEvent` object.
- **width**
Sets the width of the dragSource container.

Attributes	M	Values	Usage
------------	---	--------	-------

Uniform Resource Name (URN)

flavours		String	<i>Taglib</i> flavours="F1" <i>Classlib</i> setFlavour (F1)
id	*	String (cs)	<i>Taglib</i> id="dropTarget" <i>Classlib</i> setId ("dropTarget")
value		String	<i>Taglib</i> value="valueOfTarget" <i>Classlib</i> setValue ("valueOfTarget")
width		Unit	<i>Taglib</i> No tag available <i>Classlib</i> setWidth ("100")


Events	M	Values	Usage
onDrop		String (cs)	<i>Taglib</i> onDrop="onDropTarget" <i>Classlib</i> setOnDrop ("onDropTarget")

Example

using the taglib

```
<hbj:dropTarget
  id="dropTarget"
  flavours="F1"
  onDrop="onDropProcessTarget"
  value="aValue"
/>
```

Result

 Austria	 Germany	 Netherlands	 Belgium	 Greece
 Portugal	 Denmark	 Ireland	 Spain	 Finland
 Italy	 Sweden	 France	 Luxembourg	 UK

Which country is in charge of the EU council right now?

Drag your selection in this box

Which country will be in charge of the EU council next?

Drag your selection in this box

3.4.2.6.10 File Upload

Definition

This control to selects a file for upload. The control generates a input field and a "Browse" button. The "Browse" button activates the file browser and allows selecting the file interactively. To use the selected file in the fileUpload control you need another control, for example, a button named "Start upload", to finally start the upload.



If you use a fileUpload control in the JSP you must set the encodingType attribute of the form control to "multipart/form-data".

Example: `<hbj:form encodingType="multipart/form-data" >`

- **id**
Identification name of the fileUpload.
- **accept**
Defines the accepted MIME type.
- **maxLength**
Defines the maximum file size in byte allowed for the upload. By default there is no limit.
- **size**

Uniform Resource Name (URN)

Defines the width of the fileUpload inputField in characters. The frame of the inputField is adjusted accordingly considering the actual text font and the design attribute.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="chooseInputFile" <i>Classlib</i> setId ("chooseInputFile")
accept		String	<i>Taglib</i> accept="text/rtf" <i>Classlib</i> setAccept ("text/rtf")
maxLength		Numeric (-1)	<i>Taglib</i> maxlength="125000" <i>Classlib</i> setMaxlength ("125000")
size		Numeric (20)	<i>Taglib</i> size="30" <i>Classlib</i> setSize ("30")

Example

using the taglib

```
<hbj:form
  encodingType="multipart/form-data">
  <hbj:fileUpload
    id="myfileupload"
    maxLength="125000"
    size="50"
  />
</hbj:form>
```

Result



In an application you usually have an additional control, usually a button, to start the upload once the file has been selected with the "Browse..." button.

Here we show what the server program has to do when the user starts the upload.

In our example we define a button with an "onClick" event and specified as "onClick"

Uniform Resource Name (URN)

event handling method "onLoadFile". The "onLoadFile" method does the upload handling.

```
public void onLoadFile(Event event) {
    FileUpload fu = (FileUpload)
        this.getComponentByName("myfileupload");

    //    this is the temporary file
    if (fu != null) {
        //    Output to the console to see size and UI.
        System.out.println(fu.getSize());
        System.out.println(fu.getUI());
        //    Get file parameters and write it to the console
        IFileParam fileParam = fu.getFile();
        System.out.println(fileParam);
        //    Get the temporary file name
        File f = fileParam.getFile();
        String fileName = fileParam.getFileName();
        //    Get the selected file name and write ti to the console
        ivSelectedFileName = fu.getFile().getSelectedFileName();
        System.out.println("selected filename: " + ivSelectedFileName);
    }
}
```

3.4.2.6.10.1 Usage & Type

File upload is a control that allows to access files on the client for uploading them to the server.

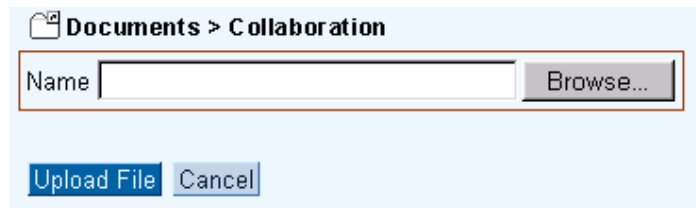


Figure 1: Example of a file upload control in a dialog window

Usage

Use the file upload control in case you want to provide the capability to pass files to the server.

Related Controls

[Breadcrumb \[Page 33\]](#), [Button \[Page 33\]](#)

3.4.2.6.10.2 Browser Support & 508

File upload is a very security sensitive control, since it allows to access the client's hard disk. For this reason, the original browser control with no design modification is used. The Browse button appears as a platform-specific standard button.

Editability in Style Editor

No editability

Accessibility - 508 Support

Not accessible.

3.4.2.6.11 Group

Definition

A framed panel to visually group controls. See also 'tray'.

- **design**

The design of the group that refers to CSS. You can select

- PRIMARYCOLOR

The panel of the group is filled with the same background as the title bar (primary color).

- SAPCOLOR

The title bar and the frame around the panel is in SAP blue and the panel has a white background.

- SECONDARYBOX

No frame around the panel. Title bar with background color (primary color), panel has a white background.

- SECONDARYBOXCOLOR

The panel is filled with a background color that is different from the title background color (primary color). No frame around the panel.

- SECONDARYCOLOR

The panel is filled with a background color that is same as the title background color (secondary color).

- **headerComponent**

Set a component for the group header.

- **id**

Identification name of the group.

- **title**

Defines the string of text placed left aligned in the title bar. If no title should be displayed an empty string (null) can be used. The width of the group is automatically adjusted to the length of the text when the 'width' attribute is set smaller than the title text width.

- **tooltip**

Uniform Resource Name (URN)

Defines the hint of the group which is displayed as the mouse cursor passes over the group, or as the mouse button is pressed but not released.

- **width**

Defines the width of the group. The width of the group is automatically adjusted to the length of the 'title'. To see an effect of the 'width' attribute 'width' has to be set higher as the width defined through the length of the 'title' string. If an empty (null) 'title' string is set no 'title' attribute is defined the width of the group is set according to the 'width' attribute.

Attributes	M	Values	Usage
design	*	PRIMARYCOLOR SAPCOLOR SECONDARYBOX SECONDARYBOXCOLOR SECONDARYCOLOR	<i>Taglib</i> design="SAPCOLOR" <i>Classlib</i> setDesign (GroupDesign.SAPCOLOR)
headerComponent		Component	<i>Taglib</i> No tag available <i>Classlib</i> setHeaderComponent (Component c)
id	*	String (cs)	<i>Taglib</i> id="Intro_Text" <i>Classlib</i> setId ("Intro_Text")
title		String	<i>Taglib</i> title="Headlines" <i>Classlib</i> setTitle ("Headlines")
tooltip		String	<i>Taglib</i> tooltip="latest news" <i>Classlib</i> setTooltip ("latest news")
width		Unit (50%)	<i>Taglib</i> width="300" <i>Classlib</i> setWidth ("300")

groupBody

Defines the items in the group. The groupBody tag has to be placed in a group tag. A groupBody can be filled with any control (checkbox, image, textView etc.).

groupHeader

Defines the header of the group. The groupHeader tag has to be placed in a group tag. A groupHeader can be filled with any control (checkbox, image, textView etc.). The control in the groupHeader is placed on the right side of the group title.

Example

using the taglib

```
<hbj:group
id="HeadlineNewsGroup"
design="SAPCOLOR"
title="latest headlines"
tooltip="all the news you need"
width="50%"
>
<hbj:groupBody>
  <hbj:textView
    encode="false"
    text="The NASDAQ on an upswing<br>Good news for homeowners"
  />
</hbj:groupBody>
</hbj:group>
```

using the classlib

```
Form form = (Form) this.getForm();
Group group = new Group();
group.setDesign(GroupDesign.SAPCOLOR);
group.setTitle("latest headlines");
group.setTooltip("all the news you need");
group.setWidth("50%");

TextView tv = new TextView("tv");
tv.setEncode(false);
tv.setText("The NASDAQ on an upswing<br>Good news for homeowners");
group.addComponent(tv);
form.addComponent(group);
```

Result

latest headlines

The NASDAQ on an upswing
Good news for homeowners

3.4.2.6.11.1 Usage & Type

A group control clusters a set of controls or information: it demonstrates which parts belong together, and separates them from other parts of content.

Uniform Resource Name (URN)

Data Sources - General Settings

ID

Name

Description

URL

☐ Data source is an SAP Portals Unifier project

Mapping Administration

Notification: You must restart the IIS in order for your changes to take effect

Authentication method

Authorization level

Teach URL

☐ Always teach

User Mapping

Update Delete

The primary and secondary group types, consisting only of a background color, can be used to highlight a part of the content .

Corporate News

Latest News:
2/5/2004
Due to unexpected oil findings on company grounds in Alaska, SAP announces extremely high revenue growth. Company to enter new business field.

General Availability of SAP SEM 3.0A
3/28/2001
The SAP SEM function of mySAP Financials has been significantly enhanced for 3.0A, specifically in the areas of business planning and simulation and the Corporate Performance Monitor.

Europe to Leapfrog U.S. in the E-Business Arena
idg.net, 3/28/2001
Less than a year ago, the skepticism of European businesses towards electronic business was much criticized. Now, with the dot-com crash in the U.S., Europeans will be able to benefit from their conservatism, according to market researcher Gartner Group "Europe is not suffering from the U.S. dot-com hangover. U.S. companies have gone the wrong road and need to back out. That will take them nine months to a year," said Alexander Drobik, vice president at Gartner EMEA (Europe, Middle East, and Africa) in interview here at the Gartner Europe Spring Symposium/ITxpo 2001.

In full-page applications, the primary and secondary group types may be used to create an area into which other controls can be placed.

Uniform Resource Name (URN)

Data Sources - General Settings Group used as an Area

ID Description

Name URL

☐ Data source is an SAP Portals Unifier project

Mapping Administration Group

Notification: You must restart the IIS in order for your changes to take effect

Authentication method

Authorization level

Teach URL

☐ Always teach

User Mapping

Update Delete

Usage

In full-page applications, use the group control to

- Group each coherent set of fields or information and separate one set from another.
- Define an area where text or controls can be placed
- Highlight a certain part of the application or information

Within an iView, there should normally be no need to highlight or separate different groups, since an iView is per definition small and simple. However, there are certain cases where it makes sense to use a group control to

- Highlight or separate a certain part of an iView to better demonstrate its structure.
For example, to show whether a certain button relates to the whole application or only to a part of it
- Highlight a certain portion of textual information within a large body of text.

General Usage Tips

Use groups only if other ways of separating information or field groups do not suffice. Group boxes look similar to the tray container of iViews and may clutter the interface visually. Preferably, use white space or vertical dividing lines to group elements, relying on the Gestalt laws.



Sometimes you don't really need groups in your iView, but simply want to create a better visual structure. Instead of misusing the group control, use the text view control to give users a better overview of your content: Create a text label for each part and add a blank line between parts to separate them.

Uniform Resource Name (URN)

Try not to nest groups; separate subgroups within groups by lines or white space. If you need to nest groups, consider nesting different group types (see below).

Positioning

[Tabstrips \[Page 33\]](#), [table views \[Page 33\]](#) and [tree views \[Page 33\]](#) are only allowed to be included in a group control if they appear together with other elements (see Layout Hierarchy for details). Placed as a single element, these controls do not need any further separation from their surroundings, as they already have distinct borders and a dominant shape.

Types

Depending on the items the group will contain, you can choose one of the offered styles. Currently, there are five group designs available, which are set using the attribute design.

Primary and Secondary Groups

The primary (design = PRIMARYCOLOR) and secondary groups (design = SECONDARYCOLOR) allow to visualize groups through a simple, flat background color. Both are suitable for textual content. Controls with a white background color, such as input fields and checkboxes, stand out well on both group designs.

Primary Group	Secondary Group
ID <input type="text"/>	ID <input type="text"/>
URL <input type="text"/>	URL <input type="text"/>
Data source is an SAP Portals Unifier project	Data source is an SAP Portals Unifier project

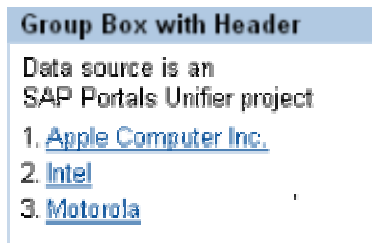
Use primary and secondary groups to:

- Define an area in a full-page application
Note: It is recommended to use the secondary (darker) group as an area background. You may then place the primary group on top of it to highlight or group parts of the area's content.
- Highlight a part of the content
- Group a set of coherent elements

Group Box

The group box design (design = SAPCOLOR) has a transparent body background. Because of its border and header bar, it has a quite dominant appearance.

Uniform Resource Name (URN)



Use the group box (figure 2) to group a set of coherent elements.

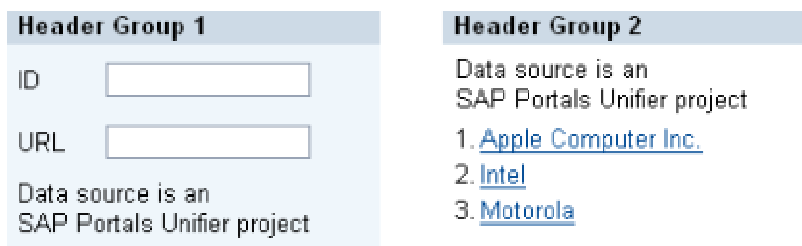
Do not nest any further groups inside a group box.

Avoid putting more than one or two of this group type adjacent to one another. They create a grid-like visual effect, which makes it hard for users to determine, which border belongs to which group.

Header Groups

Header group 1 is best suited for forms; its light body background color lets white input fields stand out well.

Header group 2 has a white body background color, which makes it unsuitable for forms. Textual content and lists work best with this group style.



Design-Relevant Attributes

The look of groups can be determined by three attributes: design selects the group type (values PRIMARYCOLOR, SAPCOLOR, SECONDARYBOX, SECONDARYBOXCOLOR, SECONDARYCOLOR), width sets the width of the group, and title sets the title text.

For details refer to page [Control API for Group \[Page 33\]](#).

Related Controls

[Tabstrip \[Page 33\]](#), [Table View \[Page 33\]](#), [Tree View \[Page 33\]](#), [Text View \[Page 33\]](#) (for headers of subgroups)

3.4.2.6.11.2 Browser Support & 508

Netscape 4.7 doesn't render the padding correctly. Title and body text begin immediately at the left edge of a group.

Exception: Body text padding in the group box with header is correct.

Uniform Resource Name (URN)

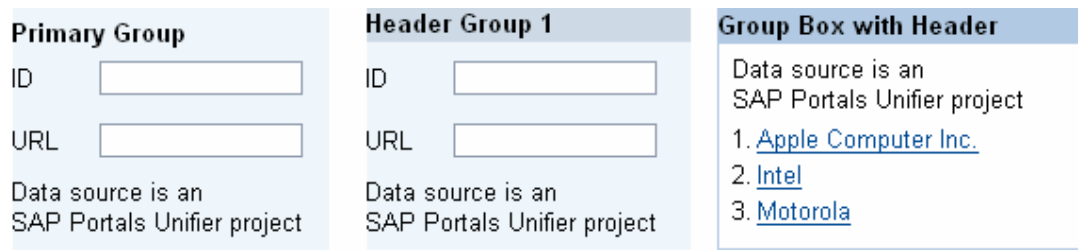


Figure 1: Examples of how groups look in some Netscape versions.

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the group control:

Group	Style	IE 5 and above	Netscape 4.7
Fonts	Font Weight of Title	x	x
Background Color	First Background Color Second Background Color Third Background Color	x x x	x x x
Borders	Border Width, Style and Color	x	x
Layout	Title Padding for Groups with Header Strip Body Padding Title Padding for Groups without Header Padding of Body Content	x x x x	
Background	Height of Container Title Background Color of Container Title Background Color of Container Body	x x x	 x x

Table 1: Editable styles for the group control

Accessibility - 508 Support

- Keyboard

The group is not inserted into the accessibility hierarchy by default. Elements inside a group have to be handled separately, depending on the respective controls included in the group.

- Default Description

Is provided by the HTMLB rendering engine.

- Application-specific Description

Set an additional description using the `setTooltip` method if needed.

Elements inside a group have to be handled separately, depending on the controls used.

Users with low vision can use the portal personalization link to select the portal's high contrast design, which was developed to offer maximum contrast. It renders the screen without using

Uniform Resource Name (URN)

many colors or shades, which leaves us with little more than black and white. Though the results may look highly unattractive to people with normal vision, sharp contrast is what users with low vision need, in order to be able to read text at all.

As a developer, you needn't do anything to enable the high contrast scheme, but you should have a feeling for what happens to your application when it is viewed in high contrast.

This is how groups look in high contrast. (Figure 2, left side). On the right hand side you can see some possible variations of group designs, which can be achieved using the Style Editor.

Primary Group	High Contrast Default Design	Primary Group	Possible Variations
Standard Text		Standard Text	
Secondary Group		Standard Text	
Standard Text			
Group Box with Header		Standard Text	
Standard Text			
Header Group 1		Standard Text	
Standard Text			
Header Group 2		Standard Text	
Standard Text			

Figure 2: Examples of groups in high contrast (left) and possible Style Editor variations (right).

3.4.2.6.12 Hover Menu

Definition

A control to create a hover menu. A hover menu works like a pop up menu in a desktop application. The hover menu can be activated on mouse click (pull down menu), on mouse over (pop up menu, help context) or on right mouse button click (context menu). The screen content under the hover menu is not destroyed. The hover menu disappears when you select an item from the menu. Every hover menu entry can have a sub hover menu. You can define a stand alone hover menu or associate a hover menu with following controls:

treeNode
textView
image

- **firstLevelVisible**
Boolean value that defines if the first level of the hovermenu is displayed.
- **id**

Uniform Resource Name (URN)

Identification name of the hoverMenu.

- **menuTrigger**

Sets the trigger mechanism which will open the hover menu.

- ONCLICK

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the left mouse button is clicked - like in a pop up menu. This option is default for web browser Netscape 4.

- ONCONTEXTMENU

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the right mouse button is clicked - like for a context menu.

- ONLRCLICK

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the right or left mouse button is clicked.

- ONMOUSEOVER

Hover menu is displayed when the mouse pointer is moved over the item linked to the hover menu. This option is default for web browsers IE5 and up and Netscape 6.

- **onHoverMenuClick**

Defines the event handling method that will be processed when the user clicks on an item in the hover menu.

- **parentItem**

Sets the parent item for the hover menu.

- **renderer**

Sets the renderer for the hover menu .

- **requiresForm**

Boolean value that defines if a form is required for the hover menu. The hoverMenu does not require a form and is not placed in one. The Javascript interface is rendered with forms which influences the ID generation.

- **standAlone**

A boolean value.

standAlone=TRUE

Creates a hover menu that is not linked to another control (image, tree node or text view). The first level of the hover menu is visible immediately without any trigger actions.

standAlone=FALSE

Creates a hover menu that has to be linked to another control (image, tree node or text view) in order to get displayed. The defined trigger method applies.

Attributes	M	Values	Usage
firstLevelVisible		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setFirstLevelVisible (false)</code>
id	*	String (cs)	<i>Taglib</i> <code>id="hover"</code> <i>Classlib</i> Id is defined by the hovermenu object itself.
menutrigger		ONCLICK (d for N4) ONCONTEXTMENU ONLRCLICK ONMOUSEOVER (d)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setMenuTrigger</code> (<code>HoverMenuTrigger.ONCLICK</code>)
parentitem			<i>Taglib</i> No tag available <i>Classlib</i> <code>setParentitem (HoverMenuItem pItem)</code>
renderer			<i>Taglib</i> No tag available <i>Classlib</i> <code>setRenderer (HoverMenuRenderer rend)</code>
requiresForm			<i>Taglib</i> <code>requiresForm="FALSE"</code> <i>Classlib</i> <code>setRequiresForm (false)</code>
standalone		FALSE (d) TRUE	<i>Taglib</i> <code>standAlone="FALSE"</code> <i>Classlib</i> No method available.

Events	M	Values	Usage
onHoverMenuClick	*	String (cs)	<i>Taglib</i> <code>onHoverMenuClick="hoverEv"</code> <i>Classlib</i> <code>setOnHoverMenuClick ("hoverEv")</code>

hoverMenuItem

Defines the items in a hoverMenu.

- altTextForImage**

Sets an alternative text for an image which is displayed as tooltip when the image is found or as text instead of the image when the image can not be found. To set the image to be displayed when the hover menu item is enabled use attribute 'imgSrc'. When the hover menu item is disabled use attribute 'disabledImgSrc'.

- checkable**

Uniform Resource Name (URN)

Sets the hover menu item as checkable. A checkable hover menu item displays a check mark on the left side of the item when the hover menu item is checked.

- **checked**

A boolean value to check (=TRUE) or uncheck (=FALSE) the hover menu item.

- **clientSideScript**

Sets a Javascript program that is started when the hover menu item is clicked.

- **disabledImgSrc**

Sets the image which is displayed when the hover menu item is disabled.

- **enabled**

A boolean value to enable (=TRUE) or disable (=FALSE) the menu item. A disabled menu item is displayed but fires no event when the item is selected (and an event handling method has been defined for the menu item).

- **id**

A string which is passed on to the event handling routine when the event occurs. A id string must be defined and must not be empty. Each hoverMenuItem must have an unique id. The id is not displayed.

- **imgSrc**

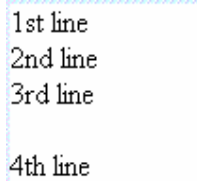
Sets the image which is displayed when the hover menu item is enabled.

- **hoverItemDivider**

A boolean value that defines a divider for actual item. Depending on the style sheets used the divider is displayed as separation line or an empty row. The divider is displayed before the visible item.

```
HoverMenu hover = new HoverMenu("hover");
HoverMenuItem item1 = hover.addMenuItem("key1", "1st line");
hover.addMenuItem(new HoverMenuItem("key2", "2nd line"));
hover.addMenuItem(new HoverMenuItem("key3", "3rd line"));
HoverMenuItem item2 = hover.addMenuItem("key4", "4th line");
item2.setHoverItemDivider(true);
```

Result



- **linkRef**

A text string that defines the URL of a page/document that will be opened when the user clicks on this item.

- **linkTarget**

Specifies the name of the frame where the document is to be opened. The following values refer to w3c HTML-standard.

- `_blank`

Uniform Resource Name (URN)

The web client should load the designated document in a new, unnamed window.

- **_self**

The web client should load the document in the same frame as the element that refers to the target.

- **_parent**

The web client should load the document into the immediate FRAMESET parent of the current frame. This value is equivalent to **_self** if the current frame has no parent.

- **_top**

The web client should load the document into the full, original window (thus canceling all other frames). This value is equivalent to **_self** if the current frame has no parent.

- **onItemClick**

Defines the event handling method that will be processed when the user clicks on this item.

- **parentMenu**

Sets the parent menu for the hover menu item.

- **subMenu**

Sets an existing hover menu as the sub hover menu of this item.

- **text**

A string that is displayed in the hover menu. A text string must be defined and must not be empty.

Attributes	M	Values	Usage
altTextForImage		String	<i>Taglib</i> No tag available <i>Classlib</i> <code>setAltTextForImage ("SAP Logo")</code>
checkable		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCheckable(true)</code>
checked		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setChecked(true)</code>
clientSideScript		String	<i>Taglib</i> <code>clientSideScript="alert('Click')"</code> <i>Classlib</i> <code>setClientSideScript ("alert('Click')")</code>
disabledImgSrc		String	<i>Taglib</i> No tag available

Uniform Resource Name (URN)

			<i>Classlib</i> setDisabledImgSrc ("disImg.gif")
enabled		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEnabled (true)
id	*	String (cs)	<i>Taglib</i> id="entry1" <i>Classlib</i> setId ("entry1")
imgSrc		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setImgSrc ("Img.gif")
hoverItemDivider		FALSE (d) TRUE	<i>Taglib</i> hoverItemDivider="TRUE" <i>Classlib</i> setHoverItemDivider (true)
linkRef		String	<i>Taglib</i> linkRef="http://www.sap.com" <i>Classlib</i> setLinkReference ("http://www.sap.com ")
linkTarget		_blank _self _parent _top	<i>Taglib</i> linkTarget="_blank" <i>Classlib</i> setLinkTarget ("_blank")
parentMenu		Component	<i>Taglib</i> No tag available <i>Classlib</i> setParentMenu (HoverMenu parent)
submenu		String (cs)	<i>Taglib</i> subMenu="sub4" <i>Classlib</i> setSubHoverMenu (HoverMenu sub)
text	*	String	<i>Taglib</i> text="First line" <i>Classlib</i> addMenuItem ("entry1", "First line")

Events	M	Values	Usage
onItemClick	*	String (cs)	<i>Taglib</i> onItemClick="onEntry1Click" <i>Classlib</i> setOnItemClick("onEntry1Click ")

Example

using the taglib

```
<hbj:hoverMenu
id="hover"
standAlone="false">
<hbj:hoverMenuItem
id="1"
text="Location"
onItemClick="itemClicked">
<hbj:hoverMenu
id="hoversub1">
<hbj:hoverMenuItem
id="111"
text="Kruger National Park"
/>
<hbj:hoverMenuItem
id="112"
text="Other African Regions"
/>
</hbj:hoverMenu>
</hbj:hoverMenuItem>
<hbj:hoverMenuItem
id="2"
text="Size"
/>
<hbj:hoverMenuItem
id="3"
text="Properties"
hoverItemDevider="true"
/>
<%
    hover.setMenuTrigger
        (com.sapportals.htmlb.enum HoverMenuTrigger.ONLRCLICK);
    %>
</hbj:hoverMenu>
<hbj:image
id="image_logo"
alt="Image not available"
hoverMenuId="hover"
src="">
<%
    IResource
    rs=componentRequest.getResource(IResource.IMAGE,
    "../mimes/rhino.gif");
    image_logo.setSrc
        (rs.getResourceInformation().getURL(componentRequest));
    %>
<hbj:image>
```

using the classlib

Uniform Resource Name (URN)

```

Form form = (Form) this.getForm();
HoverMenu hover = new HoverMenu("hover");
hover.setMenuTrigger
    (com.sapportals.htmlb.enum.HoverMenuTrigger.ONLRCLICK);
HoverMenuItem item1 = hover.addMenuItem("1", "Location");
HoverMenu item1s = item1.addSubHoverMenu("sub1");

HoverMenuItem item111 = item1s.addMenuItem
    ("111", "Kruger National Park");
HoverMenuItem item112 = item1s.addMenuItem
    ("112", "Other African Regions");

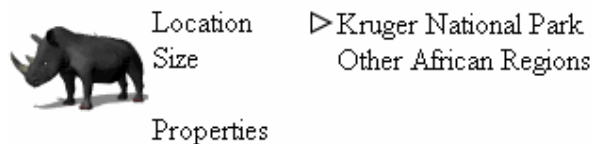
HoverMenuItem item2 = hover.addMenuItem("2", "Size");
HoverMenuItem item3 = hover.addMenuItem("3", "Properties");
item3.setHoverItemDivider(true);

Image image =
    new Image(
        request.getPublicResourcePath() + "../mimes/rhino.gif",
        "picture rhino.gif");
image.setHoverMenu(hover);

form.addComponent(hover);

```

Result (when the left or right mouse button has been clicked on the image)



3.4.2.6.13 HTML Edit

Definition

A multiline region for displaying and editing text - similar to control textEdit. The difference to textEdit is, that the text in the control is not restricted to a single font, size and style. The htmlEdit control is a WYSIWYG editor that produces HTML documents. Formatting options include:

- Bold, italic and underlined text.
- Bulleted and numbered lists.
- Text fonts, text size and text color can be selected.
- Links and images can be added.

This control is available for web browser Internet Explorer Version 5.x and higher.

- **id**
Identification name of the htmlEdit control.
- **doAlign**

Uniform Resource Name (URN)

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Alignment" functions - left, center and right Alignment.

- **doBackground**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Background" function for text.

- **doBold**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Bold-Characters" functions.

- **doCutCopyPaste**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Cut, Copy and Paste" functions.

- **doForeground**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Foreground" function for text (text color).

- **doImage**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Insert Image" function.

- **doInOutdent**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Increase Indent/Decrease Indent" function.

- **doItalic**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Italic-Characters" function.

- **doLink**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Insert Link to the Web" function.

- **doLinkKM**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Insert Link to the Knowledge Management (KM) " function.

- **doList**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Bulleted/Numbered List" functions.

- **doPreview**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the document "Preview" function.

- **doPrint**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the document "Print" function.

- **doStrikethrough**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Strikethrough" function.

- **doStyle**

Uniform Resource Name (URN)

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Style" function.

- **doSuperSubScript**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Super/Sub script" function.

- **doTextsize**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Textsize" function.

- **doUnderline**

A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Underline" function.

- **height**

Defines the height of the htmlEdit control.

- **imageHeight**

Defines the height of the image to be inserted.

- **imageSrc**

Defines the name of the image to be inserted.

- **imageTooltip**

Defines the tooltip for the image.

- **imageWidth**

Defines the width of the image to be inserted.

- **linkTarget**

Defines the target of the link to be inserted. The following values refer to w3c HTML-standard.

- **_blank**

The web client should load the designated document in a new, unnamed window.

- **_self**

The web client should load the document in the same frame as the element that refers to the target.

- **_parent**

The web client should load the document into the immediate FRAMESET parent of the current frame. This value is equivalent to **_self** if the current frame has no parent.

- **_top**

The web client should load the document into the full, original window (thus canceling all other frames). This value is equivalent to **_self** if the current frame has no parent.

- **linkText**

Defines the text of the link to be inserted.

- **linkURL**

Defines the URL (href=) of the link to be inserted.

Uniform Resource Name (URN)

- **oldId**
Defines the oldId attribute for the htmlEdit control.
- **oldText**
Defines the previous content of the control. The previous content is the content before you start an "insert image" or "insert link" function.
- **onInsertImage**
Defines the event handling method that will be processed when the user activates the "Insert Image" function.
- **onInsertLink**
Defines the event handling method that will be processed when the user activates the "Insert Link" function.
- **text**
Defines the string of text displayed. This text can be edited and/or new text can be added.
- **width**
Defines the width of the htmlEdit control.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="htmlEdit" <i>Classlib</i> setId ("htmlEdit")
doAlign		FALSE (d) TRUE	<i>Taglib</i> doAlign="TRUE" <i>Classlib</i> setDoAlign (true)
doBackground		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoBackground (true)
doBold		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoBold (true)
doCutCopyPaste		FALSE (d) TRUE	<i>Taglib</i> doCutCopyPaste"TRUE" <i>Classlib</i> setDoCutCopyPaste (true)
doForeground		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoForeground (true)
doImage		FALSE (d)	<i>Taglib</i>

Uniform Resource Name (URN)

		TRUE	doImageLink="TRUE" <i>Classlib</i> setDoImage (true)
doInOutDent		FALSE (d) TRUE	<i>Taglib</i> doInOutDent="TRUE" <i>Classlib</i> setDoInOutDent (true)
doItalic		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoItalic (true)
doLink		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoLink (true)
doLinkKM		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoLinkKM (true)
doList		FALSE (d) TRUE	<i>Taglib</i> doList="TRUE" <i>Classlib</i> setDoList (true)
doPreview		FALSE (d) TRUE	<i>Taglib</i> doPreview="TRUE" <i>Classlib</i> setDoPreview (true)
doPrint		FALSE (d) TRUE	<i>Taglib</i> doAlign="TRUE" <i>Classlib</i> setDoAlign (true)
doStandardEdit		FALSE (d) TRUE	<i>Taglib</i> doStandardEdit="TRUE" <i>Classlib</i> No method available
doStrikethrough		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoStrikethrough (true)
doStyle		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoStyle (true)
doSuperSubScript		FALSE (d) TRUE	<i>Taglib</i> No tag available

Uniform Resource Name (URN)

			<i>Classlib</i> setDoSuperSubScript(true)
doTextsize		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoTextsizetrue)
doUnderline		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setDoUnderline(true)
height		Unit	<i>Taglib</i> height="300px" <i>Classlib</i> setHeight("300px")
imageHeight		Unit	<i>Taglib</i> No tag available <i>Classlib</i> setImageHeight("100px")
imageSrc		String	<i>Taglib</i> No tag available <i>Classlib</i> setImageSrc("SAPLogo.gif")
imageTooltip		String	<i>Taglib</i> No tag available <i>Classlib</i> setImageToolTip("SAP Logo")
imageWidth		Unit	<i>Taglib</i> No tag available <i>Classlib</i> setImageWidth("100px")
linkTarget		_blank _self _parent _top	<i>Taglib</i> No tag available <i>Classlib</i> setLinkTarget("_top")
linkText		String	<i>Taglib</i> No tag available <i>Classlib</i> setLinkText("Go to SAP")
linkURL		String	<i>Taglib</i> No tag available <i>Classlib</i> setLinkURL("http://www.sap.com")
oldId		String	<i>Taglib</i> No tag available <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setOldId("prevID")</code>
oldText		String	<i>Taglib</i> No tag available <i>Classlib</i> <code>setOldText("Episode I")</code>
text		String	<i>Taglib</i> <code>text="The Star Wars Saga"</code> <i>Classlib</i> <code>setText("The Star Wars Saga")</code>
width		Unit	<i>Taglib</i> <code>width="400px"</code> <i>Classlib</i> <code>setWidth("400px")</code>

Events	M	Values	Usage
onInsertImage		String (cs)	<i>Taglib</i> <code>oninsertImage="onImage"</code> <i>Classlib</i> <code>setOninsertImage("onImage")</code>
onInsertLink		String (cs)	<i>Taglib</i> <code>onInsertLink="onLink"</code> <i>Classlib</i> <code>setOnInsertLink="onLink"</code>

Example

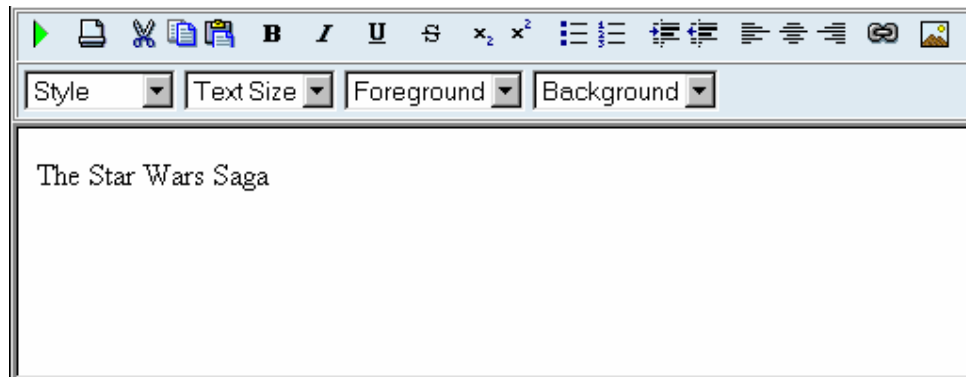
using the taglib

```
<hbj:htmlEdit
  id="htmlEdit"
  text="The Star Wars Saga"
  height="200"
  width="300"
  doPrint="TRUE"
  doPreview="TRUE"
  doCutCopyPaste="TRUE"
  doList="TRUE"
  doAlign="TRUE"
  doInOutdent="TRUE"
  onInsertLink="onLink"
  onInsertImage="onImage"
/>
```

using the classlib

Uniform Resource Name (URN)

```
Form form = (Form) this.getForm();
HtmlEdit he = new HtmlEdit("myNavigation");
he.setText("The Star Wars Saga");
he.setHeight("200");
he.setWidth("300");
he.setDoPrint(true);
he.setDoPreview(true);
he.setDoCutCopyPaste(true);
he.setDoList(true);
he.setDoAlign(true);
he.setDoInOutdent(true);
he.setOnInsertLink("onLink");
he.setOnInsertImage("onImage");
form.addComponent(he);
```

Result**3.4.2.6.14 Image****Definition**

Displays a bitmap in GIF or JPEG format. The width and height of the image can be specified.

- **alt**

Defines an alternative text for the 'src' attribute. If the 'src' bitmap cannot be found or opened (for example, unrecognized graphic format) the 'alt' text is displayed with a red X in front of it and surrounded by a frame indicating the bitmap size.

If no 'tooltip' attribute is set for the image the alternative text is displayed as the mouse cursor passes over the image, or as the mouse button is pressed but not released.

- **height**

Defines the height of the bitmap. If 'height' is omitted the height of the image is determined by the bitmap itself.

- **hoverMenuId**

Defines which hover menu is displayed for this tree node. You can define different trigger methods to display the hover menu. For more details, see hover menu.

Uniform Resource Name (URN)

- **id**

Identification name of the image.

- **imageMapId**

Identification name of the imageMap control that is associated with the image. The imageMap control allows you to define areas on the image that are clickable. The areas can be associated with a link or an event. For more details on imageMap, see imageMap control description.

- **src**

Name of bitmap (=source). The name of the bitmap is case sensitive. To have access to MIME delivered with the portal see "accessing MIME". The images are usually stored in the public resource path of the component in the subfolder /images. The public resource path can be determined with the JSP command:

```
<% String PublicURL = componentRequest.getPublicResourcePath();
%>
```

You can also get the complete path to image include the subfolder /mimes with the command:

```
<% String ImageURL = componentRequest.getPublicResourcePath() +
    "../mimes/"; %>
```



We use the string variable ImageURL in following table to demonstrate the setting of the src attribute.

Another possibility is to get the full URL of the image and set the src attribute in a scriptlet.

```
<%
IResource rs = componentRequest.getResource(IResource.IMAGE,
    "../mimes/mypicture.gif");
image.setSrc(rs.getResourceInformation().getURL(componentRequest
));
%>
```

- **tooltip**

Defines the hint of the image which is displayed as the mouse cursor passes over the image, or as the mouse button is pressed but not released.

- **width**

Defines the width of the bitmap. If 'width' is omitted the width of the image is determined by the bitmap itself.

Attributes	M	Values	Usage
alt	*	String	<i>Taglib</i> alt="Walldorf picture"

Uniform Resource Name (URN)

			<i>Classlib</i> setAlt ("Walldorf picture")
height		Unit	<i>Taglib</i> height="150" <i>Classlib</i> setHeight="150"
hoverMenuId		String	<i>Taglib</i> hoverMenuId="imgHover1" <i>Classlib</i> setHoverMenu (HoverMenu menu)
id	*	String (cs)	<i>Taglib</i> id="Hometown" <i>Classlib</i> setId ("Hometown")
imageMapId		String (cs)	<i>Taglib</i> imageMapId="imageMap" <i>Classlib</i> setImageMap ("imageMap")
src	*	String (cs)	<i>Taglib</i> src="<%=ImageUrl+\ "wdf.jpdp\ " %>" <i>Classlib</i> setSrc (ImageUrl + "wdf.jpg")
tooltip		String	<i>Taglib</i> tooltip="center of ebiz" <i>Classlib</i> setTooltip ("center of ebiz")
width		Unit	<i>Taglib</i> width="300" <i>Classlib</i> setWidth ("300")

Example**using the taglib**

```
<hbj:image
  id="Logo"
  tooltip="center of ebiz"
  width="70"
  height="35"
  alt="picture saplogo.gif"
  src="<%= ImageURL + \"saplogo.gif\" %>
/>
```

using the classlib

Uniform Resource Name (URN)

```
IPortalComponentRequest request =  
    (IPortalComponentRequest) this.getRequest();  
  
Form form = (Form) this.getForm();  
Image image =  
    new Image(  
        request.getWebResourcePath() + "../mimes/saplogo.gif",  
        "picture saplogo.gif");  
image.setTooltip("center of ebiz");  
image.setWidth("70");  
image.setHeight("35");  
form.addComponent(image);
```

Result**3.4.2.6.14.1 Usage & Type**

The image control displays a bitmap GIF or JPEG format. The width and height of the image can be specified.



Figure 1: Example of an image in an iView. The image appears only after the user makes a selection via the shuffler placed above the image.

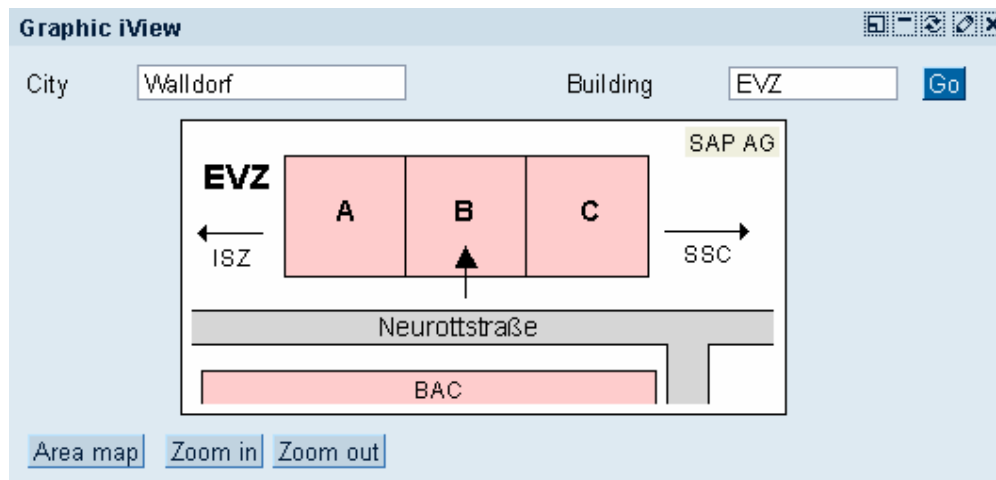


Figure 2: Example of an image with a row of image-related function buttons

Usage

The image control displays a bitmap in GIF or JPEG format. The width and height of the image can be specified.

Photographs, graphics, charts and diagrams, maps, sketches, animated graphics and video (movies) may be used in the portal. If used properly, they carry great amounts of information, which would take up much more time and screen-space, if they were explained in words.



Although icons are also images, they are not allowed in iViews, except for displaying status information. That is, there are no function icons on buttons or tabs. For more information on status icons in the portal, see the SAP Reference Lists in the SAP Design Guild.

Interacting with Images

- If graphics and data can be selected from several sets, or if the amount of data has to be reduced, place a filter or shuffler above the image. (See figure 1, above)
- Place buttons for image-related functionality and status information (for example, zoom factor) below the image and left align them. (See figure 2, above)
- Place buttons related to the whole iView in the lower left corner; these buttons may reside in the same row as the table-related buttons.
- If there is an emphasized button, it is the leftmost button of the respective button group (image-related or iView-related). There must not be more than one emphasized button in an iView.

Legend

Always provide an appropriate legend. Place legends or other text below the image or to its right, depending on the format of the image and the iView or page layout.

Tips for Using Images

- Align graphics so that their main contents points towards the text, not away from it.
- Crop graphics to the relevant section; make them as small as possible and avoid irrelevant and distracting elements.

Example: Do not show a US map if you want to illustrate data in Michigan - use a Michigan map instead.

- Use high quality graphics.

Example: Do not draw graphics by yourself, involve graphic designers.

- Care for the correct format of images:

JPEG for photos and images with many colors and gradations.

GIF for images with flat-colored areas and bold lines, like diagrams or cartoon, and images with (less than) 256 colors. Typically, screen dumps work better in GIF format.

Sharp-edged graphics work well as transparent GIF, on any background. Using transparent GIF format for round, smooth forms and large-sized text, as used in many logos, may cause problems if you don't know what color their background will be.

Types

Charts

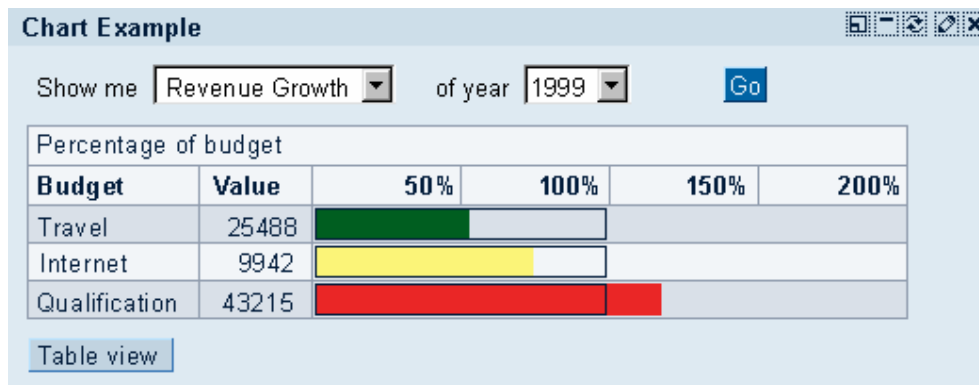


Figure 3: A horizontal bar chart table; for exact comparisons, the values have been added to the chart.

See Chart for details and Recommendations for Charts and Graphics in the SAP Design Guild for thorough information on the usage of charts.

Animated Graphics, Video, or Movie



Figure 4: Animation can explain procedures or be just fun to watch...

The image control currently supports only animated GIFs.

Sketches

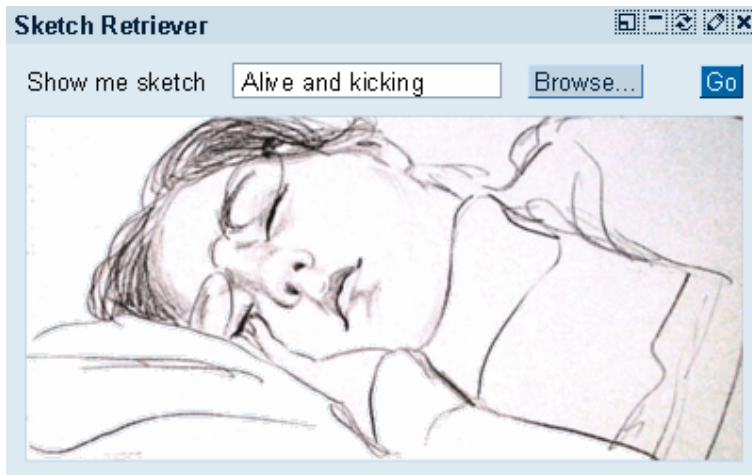


Figure 5: A sketch may be effective for fast communication or serve as a preliminary version of a diagram

Design-relevant Attributes

You can set the height (height) and width (width) of an image, also the tooltip text (tooltip), which is displayed as the mouse cursor passes over the image, or as the mouse button is pressed but not released.

Usage - Height and Width

Do not scale images in the browser by changing the values for height and width. This results in poor image quality.

Related Controls

[Table View \[Page 33\]](#)

3.4.2.6.14.2 Browser Support & 508

The image control is compatible with all browser types.

Editability in Style Editor

Customers can customize portal images used within the portal's outer frame and control-rendering (iView function images, table buttons, etc.) quite easily via Style Editor. The tool offers no editable styles related to images placed as portal content.

Accessibility - 508 Support

- **Keyboard**
Images are not inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed. Do not use the `setAlt` method that sets the alternate text (alt attribute).

3.4.2.6.15 Input Field

Definition

An framed area that allows user input. The `inputField` can be displayed with default input. A user can type new text or edit the existing text. The `inputField` control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **BCD**
Defines the value of the input field and the 'type' as data type Float (BCD = Binary Coded Decimal). The value has to match the data type format. This attribute combines the attributes 'type' and 'value'. See description of the 'type' attribute for more details how the `inputField` handles types.
- **date**
Defines the value of the input field and the 'type' as data type Date. The value has to match the data type format according to the locale setting. This attribute combines the attributes 'type' and 'value'. See description of the 'type' attribute for more details how the `inputField` handles types.
- **design**
Defines the size of the input field. The value for this attribute can be "STANDARD" or "SMALL".
- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).

Uniform Resource Name (URN)

A boolean value that defines if the inputField allows input. A disabled (enabled = false) inputField has a different background color.

- **firstDayOfWeek** - *Deprecated*

The "first day of the week" that is displayed in the datenavigator (when you use input field "type=DATE" and "showHelp=true") is now controlled by the Java calendar class. The "firstDayOfWeek" attribute has no more effect on the datenavigator.

- **id**

Identification name of the inputField.

- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the inputField component.

- **integer**

Defines the value of the input field and the 'type' as data type Integer. The value has to match the data type format. This attribute combines the attributes 'type' and 'value'. See description of the 'type' attribute for more details how the inputField handles types.

- **maxlength**

Defines the maximum amount of characters allowed for the inputField. If the type attribute is set for example, to date or time the 'maxlength' has to take care of the characters delivered by this format and local settings.

- **password**

A boolean value that controls the echo of the inputField. If set to "true" the typed in characters are echoed - displayed as asterisks (*). A common use for this attribute is to inquire passwords.

- **required**

This attribute sets a different style sheet class for a required inputField (that is a inputField that has to be filled out by the user). That gives you the opportunity do create a complete different look for a required inputfield (for example, light blue background). Use the 'label' control to indicate with a character that the inputField is required (for example, an asterisk).

- **showHelp**

A boolean value that activates a help button when set to "true". 'showHelp' shows effect only when 'type' is set to "date". The help button pops up the date navigator allowing selection of the date by clicking on to the required day.

If you define a textView before the inputfield (to explain the meaning of the inputField) and than an inputField with enabled showHelp it is recommended to place the textView and the inputField in a grid or a tableView for better formatting.

If textView is not placed in a grid or tableView a line wrap between textView and inputField will occur.

- **size**

Defines the width of the inputField in characters. The frame of the inputField is adjusted accordingly considering the actual text font and the design attribute.

The inputField width can also be set by the attribute 'width'. If 'size' and 'width' are set the 'width' attribute has priority and overwrites the 'size' setting.

- **tooltip**

Uniform Resource Name (URN)

Defines the hint of the inputField which is displayed as the mouse cursor passes over the inputField, or as the mouse button is pressed but not released.

- **time**

Defines the value of the input field and the 'type' as data type Time. The value has to match the data type format according to the locale setting. This attribute combines the attributes 'type' and 'value'. See description of the 'type' attribute for more details how the inputField handles types.

- **type**

If 'type' is set to date a help button to call the dateNavigator can be generated (see 'showHelp'). For all other types for the inputField, the frame of the inputField is displayed in red, if the value does not match the type (for example, 'type' is set to Integer and 'value' is "ABC").

- **valid**

A boolean value that controls the frame of the inputField.

invalid="true": InputField is displayed with a red frame.

invalid="false": InputField is displayed with the regular frame color

- **value**

Default string that is displayed in the inputField frame. The 'maxlength' attribute has no effect on the 'value' attribute. The 'value' string is not truncated to 'maxlength'. By default the value is handled as data type string. For other data types set the 'type' attribute or use the BCD, Date, Integer or Time methods to set the value.

- **visible**

A boolean value that defines if the inputField is visible or invisible.

- **width**

Defines the width of the inputField in pixel or percent. This attribute allows better adjustment of the inputField in a form.

The inputField width can also be set by the attribute 'width'. If 'size' and 'width' are set the 'width' attribute has priority and overwrites the 'size' setting.

Attributes	M	Values	Usage
BCD		String	<i>Taglib</i> No tag available <i>Classlib</i> setBCD ("1.15")
date		String	<i>Taglib</i> No tag available <i>Classlib</i> setDate ("3.12.2003")
design		STANDARD (d) SMALL	<i>Taglib</i> design="SMALL" <i>Classlib</i> setDesign ("SMALL")
enabled*		FALSE TRUE (d)	<i>Taglib</i> disabled="TRUE"

Uniform Resource Name (URN)

			<i>Classlib</i> setEnabled(false)
id	*	String (cs)	<i>Taglib</i> id="GetInput" <i>Classlib</i> setId("GetInput")
jsObjectNeeded**		FALSE (d) TRUE	<i>Taglib</i> jsObjectNeeded="TRUE" <i>Classlib</i> setJsObjectNeeded(true)
integer			<i>Taglib</i> No tag available <i>Classlib</i> setInteger("10")
maxlength			<i>Taglib</i> maxlength="25" <i>Classlib</i> setMaxlength(25)
password		FALSE (d) TRUE	<i>Taglib</i> password="TRUE" <i>Classlib</i> setPassword(true)
required		FALSE (d) TRUE	<i>Taglib</i> required="TRUE" <i>Classlib</i> setRequired(true)
showHelp takes effect only if 'type' is set to "date".		FALSE (d) TRUE	<i>Taglib</i> showHelp="TRUE" <i>Classlib</i> setShowHelp(true)
size		Numeric (30)	<i>Taglib</i> size="35" <i>Classlib</i> setSize("35")
time		String	<i>Taglib</i> No tag available <i>Classlib</i> setTime("00:14:01")
tooltip		String	<i>Taglib</i> No tag available <i>Classlib</i> setTooltip("Order number")
type		BCD BOOLEAN DATE	<i>Taglib</i> type="INTEGER" <i>Classlib</i>

Uniform Resource Name (URN)

		INTEGER STRING TIME	setType(DataType.INTEGER)
valid		FALSE TRUE (d)	<i>Taglib</i> invalid="TRUE" <i>Classlib</i> setValid(false)
value		String	<i>Taglib</i> value="Your name here" <i>Classlib</i> setValue("Your name here")
visible		FALSE TRUE (d)	<i>Taglib</i> visible="false" <i>Classlib</i> setVisible(false)
width		Unit	<i>Taglib</i> width="200" <i>Classlib</i> setWidth("200")

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the [Component \[Page 33\]](#) component.

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Example

using the taglib

```
<hbj:inputField
  id="InputName"
  type="string"
  maxlength="100"
  value="Your name here"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
InputField input = new InputField("InputName");
input.setType(DataType.STRING);
input.setMaxLength(100);
input.setValue("Your name here");
form.addComponent(input);
```


Result

3.4.2.6.15.1 Usage & Type

Input fields are used for entering and displaying data in forms. The data can be of various types, such as Date, Integer, or String.

Input fields can have different behaviors, such as password, read-only, or required, and different states, such as normal and error.

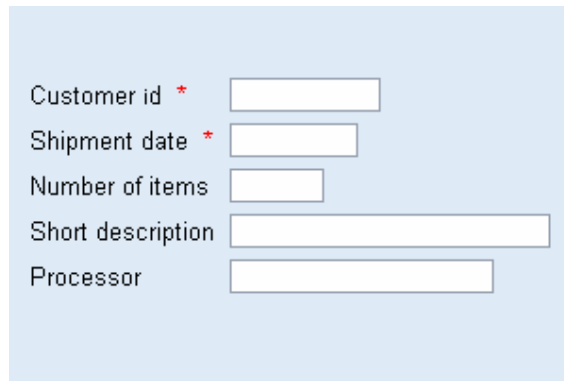


Figure 1: Example of grouped input fields with and without required inputs

Usage

In most cases, input fields appear in combination with the label control and sometimes with additional elements, such as descriptions or buttons.



Figure 2: Input field with label (left) and additional elements

Typically, the label is placed left to the input field, while the description follows the field. There is one exception to this rule: You may use small labels if you place the labels above input fields to achieve a more compact design (figure 3).

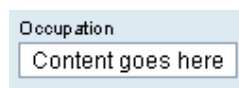


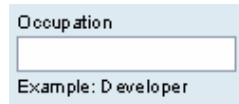
Figure 3: Use small labels for labels that are placed above their associated input fields

In addition, help texts can be placed right to the field below it.

Uniform Resource Name (URN)

Help Texts

Help texts are special descriptions that are placed behind the input field or - if space is limited - below the input field and left-aligned with it (figure 4). Do not use the label control for help texts. Use the text view control, instead. Set the help text size to small (style Legend).



Occupation

Example: Developer

Figure 4: A small label used in conjunction with a small help text below the associated input fields - do not use the label control for the help text (below the input field)

Width and Alignment

Often input fields are grouped to form a semantic block of input data, such as address data, or bank data. In that case, input fields should indicate how many characters the user has to enter. Therefore, it is not appropriate to set all fields within a group to the same size. That is important because input fields are often used in combination with other input types, such as checkboxes and radio buttons.



First name *

Last name *

Gender

City

Street

☐ List my mail address for free

First name *

Last name *

Gender

City

Street

☐ List my mail address for free

Figure 5: Example of grouped input fields with the width attribute set to appropriate values

Use the [grid layout \[Page 33\]](#) control for aligning fields and labels. Both fields and labels are left-aligned within the grid. The number of characters of the longest label determines the offset between the labels and the input fields.

Uniform Resource Name (URN)

Types

Input fields come in two sizes: standard size and small size. They are set using the attribute size to STANDARD (default) or SMALL.

Usage - Sizes

Usually, only the standard size is used. If the screen real estate is limited, the small size might be appropriate. Do not mix small and standard input fields within one field group.

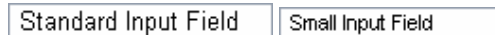


Figure 6: Standard input field (left) and small input field (right)

Design-relevant Attributes

A number of attributes allow to display several different states and behaviors of an input field, such as read-only, error, password field and required field.

Read-only Input Field	<input type="text" value="You can't change me!"/>
Invalid Input Field	<input type="text" value="Invalid Value"/>
Password Input Field	<input type="password" value="*****"/>
Required Input Field *	<input type="text" value="You have to enter this"/>

Figure 7: Different states of input fields

These states and behaviors are set by assigning the value TRUE to the Boolean attributes disabled, invalid, password, and required.

The data type of input fields, such as Integer, String, etc., is set using the attribute type. For possible values and further attributes, see page Control API for Input Field.

In addition, there are specialized input fields, which may also be accompanied by a value help. Below, we present the date field as example (this is currently the only type of value help that is supported - see page [Control API for Input Field \[Page 33\]](#) for details).

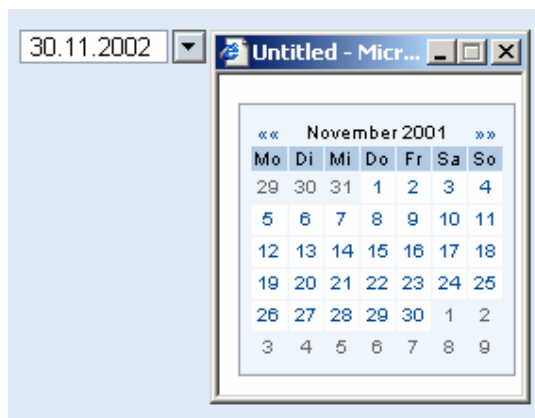


Figure 8: Date input field with date picker

Uniform Resource Name (URN)

Usage - Required Fields

Some fields require that users enter a value before they can continue, for example, before they can save data. Set `required = TRUE` for making an input field a required field; its corresponding label has also to be set to required.

Usage - Read-only Fields

Read-only fields (`disabled = TRUE`) are input fields that do not allow users to enter data.

Use read-only fields for data that have previously been entered by the user, for example, on a preceding page or during a previous session, or by the system, and that currently cannot be changed by the user. Often, the data that the user is allowed to enter depend on the protected data that are displayed in a read-only field.

Usage - Password Fields

Use password fields (`password = TRUE`) when users have to enter a password, for example, in a login dialog.

Usage - Invalid Fields (Error State)

Set the error state for an input field (`invalid = TRUE`), whenever the system detects an input error that the user committed. Depending on the system behavior, an additional error message should appear immediately, or after a certain "stable" system state has been reached, when the system performs a more thorough error check.

Related Controls

[Label \[Page 33\]](#), [Checkbox \[Page 33\]](#), [Radio Button \[Page 33\]](#), [Grid Layout \[Page 33\]](#), [Flow Layout \[Page 33\]](#), [Group \[Page 33\]](#)

3.4.2.6.15.2 Browser Support & 508

Renders in every supported browser.

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the input field control:

Group	Style	IE 5 and higher	Netscape 4.7
Field Styles	Border Width, Style and Color Padding	x x	
Standard-sized Field	Font Size Height	x x	x
Small-sized Field	Small Font Size	x	

Uniform Resource Name (URN)

	Small Height	x	x
Invalid Field	Border for Invalid Input	x	
Required Field	Font Color of "Required" Indicator	x	x
Background	Background Color of Editable Fields	x	
	Background Color of Non-Editable Fields	x	

Table 1: Editable styles for the input field control

Accessibility - 508 Support

Input fields have to be used in combination with the label element which points to the assigned input field. This ensures, that screenreaders are aware of the relationship between the both elements and can read the correct label to the according field.

- **Keyboard**
The input field is inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed.
- **Label**
Has to be connected to a label control (use method `setLabelFor` for identifying the corresponding input field).

3.4.2.6.16 Isolated HTML Container

Definition

The `IsolatedHtmlContainer` control displays a HTML document represented by its URL, inside an `iFrame`.



The `IsolatedHtmlContainer` cannot handle HTML documents which contain for example, JavaScript that places the document in a self defined frame. These HTML documents will use the entire web browser window, regardless of the `IsolatedHtmlContainer`.

This control is available for web browser Internet Explorer and Netscape 6.0 and higher. In Netscape 4.7 the HTML document will be rendered in a new browser window, similar to the 'target=_blank' argument in the "link" control.

- **bordered**
A boolean value that, if set to true, draws a border around the `IsolatedHtmlContainer` control.

Uniform Resource Name (URN)

- **height**

Specifies the height of the IsolatedHtmlContainer control.

- **id**

Identification name of the IsolatedHtmlContainer control.

- **scrolling**

Defines if the IsolatedHtmlContainer control has scroll bars.

- AUTO

Displays scroll bars only if the HTML document exceeds the specified height and width of the IsolatedHtmlContainer control.

- YES

Displays the IsolatedHtmlContainer control always with scroll bars.

- NO

Displays no scroll bars regardless of the size of the HTML document. The portion of the HTML document that exceeds the specified height and width of the IsolatedHtmlContainer control is clipped.

- **srcUrl**

Specifies the address of the page/document to be displayed in the IsolatedHtmlContainer control.

- **tooltip**

Defines the hint of the link which is displayed as the mouse cursor passes over the IsolatedHtmlContainer, or as the mouse button is pressed but not released.

- **width**

Specifies the width of the IsolatedHtmlContainer control.

Attributes	M	Values	Usage
bordered		FALSE (cs) TRUE	<i>Taglib</i> bordered="TRUE" <i>Classlib</i> setBordered(true)
height		Unit	<i>Taglib</i> height="300" <i>Classlib</i> setHeight ("300")
id	*	String (cs)	<i>Taglib</i> id="isohtmlcont" <i>Classlib</i> setId("isohtmlcont")
scrolling		AUTO YES NO	<i>Taglib</i> scrolling="NO" <i>Classlib</i> setScrolling (Scrolling.NO)
srcUrl	*	String	<i>Taglib</i>

Uniform Resource Name (URN)

			srcUrl="http://www.sap.com" <i>Classlib</i> setSrcUrl("http://www.sap.com")
tooltip		String	<i>Taglib</i> tooltip="Center of ebiz" <i>Classlib</i> setTooltip("Center of ebiz")
width		Unit	<i>Taglib</i> width="500" <i>Classlib</i> setWidth("500")

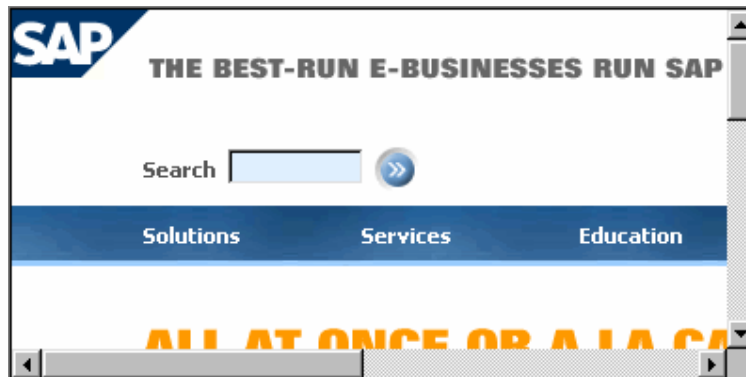
Example

using the taglib

```
<hbj:isolatedHtmlContainer
  id="isohtmlCont"
  width="400"
  height="200"
  srcUrl="http://www.sap.com"
  scrolling="AUTO"
  bordered="true"
  tooltip="An isolated container"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
IsolatedHtmlContainter ihc = new IsolatedHtmlContainer("isohtmlCont");
ihc.setWidth("400");
ihc.setHeight("200");
ihc.setSrcUrl("http://www.sap.com");
ihc.setScrolling(Scrolling.AUTO);
ihc.setBordered(true);
ihc.setTooltip("An isolated container");
form.addComponent(ihc);
```

Result**3.4.2.6.17 Item List****Definition**

The item list can be displayed as bulleted or numbered (ordered) list. An element in an itemList is called listItem.

- **bulletUri**
Sets the image that is displayed as bullet in the item list.
- **id**
Identification name of the itemList.
- **ordered**
A boolean value that defines if the list is displayed as bulleted (ordered="false") or numbered list (ordered="true").

Attributes	M	Values	Usage
bulletUri		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setBulletUri(ImageUri + "arrow.gif")</code>
id	*	String (cs)	<i>Taglib</i> <code>id="ImportantItems"</code> <i>Classlib</i> <code>setId("ImportantItems")</code>
ordered		FALSE (d) TRUE	<i>Taglib</i> <code>ordered="TRUE"</code> <i>Classlib</i> <code>setOrdered(true)</code>

Uniform Resource Name (URN)

listItem

Defines the items in the itemList. A listItem can be built with any control (checkbox, image, textView etc.) and one listItem can contain more then one control.

Example

using the taglib

```
<hbj:itemList
  id="ImportantItems"
  ordered="true"
>
  <hbj:listItem>
    <hbj:textView
      text="Introduction"
    />
  </hbj:listItem>
  <hbj:listItem>
    <hbj:textView
      text="Definitions"
    />
  </hbj:listItem>
  <hbj:listItem>
    <hbj:textView
      text="Main Part"
    />
  </hbj:listItem>
  <hbj:listItem>
    <hbj:textView
      text="Conclusion"
    />
  </hbj:listItem>
</hbj:itemList>
```

using the classlib

```
ItemList iteml = new ItemList();
iteml.setOrdered(true);
iteml.addText("Introduction");
iteml.addText("Definitions");
iteml.addText("Main Part");
iteml.addText("Conclusion");
form.addComponent(iteml);
```

Result

1. Introduction
2. Definitions
3. Main Part
4. Conclusion

3.4.2.6.17.1 Usage & Type

Use an ordered item list if you want to represent items in a certain predefined sequence, for example, a ranking. Use an unordered list if there is no predefined item order.



Figure 1a-b: Ordered item list (left) and unordered item list (right)

See also [List Box – Usage & Type \[Page 33\]](#) for details when to use dropdown list box, list box, item list and table view.

Usage

Use item lists whenever you want to present a list of items in an unobtrusive way, where reading is the primary usage and where there is - apart from links - no interaction on the list elements.

Lists have the following characteristics:

- List items are read-only.
- List items may contain links.
- Item lists do not scroll. Therefore make sure that all list items fit the iView!
- Item lists consist of one column only; for multiple columns include the lists in an HTML table with one row and several columns (2-3) or use a table view.
- Lists may be ordered (numbers) or unordered (bullets).
- Lists may be nested; do not use more than 2-3 levels!

Description - Label or Heading

Item lists may have a label or heading. A descriptive label may be placed above or to the left of the item list, a heading should typically be placed above the item list.

Use the label control for both labels and headings. Note that the label control allows to set font attributes in order to emphasize headings.

Uniform Resource Name (URN)



Figure 2: Item list with a heading

Types

There are two types of item lists, an unordered or bullet list, and an ordered or numbered list. Both types are set using the Boolean attribute `ordered`: `ordered = TRUE` renders a numbered list, `ordered = FALSE` a bullet list.

Usage - Types

Use an ordered item list if you want to present items in a certain predefined sequence, for example, a ranking. Use an unordered list if there is no predefined item order.

Related Controls

[Table View \[Page 33\]](#), [Text View \[Page 33\]](#), [Link \[Page 33\]](#), [Listbox \[Page 33\]](#), [Label \[Page 33\]](#)

3.4.2.6.17.2 Browser Support & 508

Supported by all browsers.

Editability in Style Editor

The following characteristics are editable in the Style Editor:

- List Style Image (List Bullet)
- List Margin

In the Style Editor, it is possible to modify the following attributes of the item list control:

Style	IE5 and above	Netscape 4.7
URL to List Style Image	x	x

List Margin	x	
-------------	---	--

Table 1: Editable styles for the item list control

Accessibility - 508 Support

- **Keyboard**
The item list is not inserted into the accessibility hierarchy by default.
- **Default Description**
Not needed.
- **Application-specific Description**
Not needed.

3.4.2.6.18 Label

Definition

A multi line region for displaying text. Text in the component is restricted to a single font, size and style unless set with HTML commands.

The label control works similar to [textView \[Page 33\]](#) control. In addition a label has an associated component like an `inputField` or `listBox`. The label control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **design**
Defines the appearance of the label. The CSS controls how the different options are rendered. The following description is based on the standard CSS delivered.
 - **LABEL**
Text size and attributes STANDARD
 - **LABELSMALL**
Text size -2 in comparison to STANDARD
- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).
A boolean value that enables (=true) or disables (=false) the label control. A disabled label has a different text color to show the user that it is disabled.
- **encode**
A boolean value that defines how the label text is interpreted. HTML text formatting commands (for example, `<h1>`, `<i>` etc.) can be used to change the display of the text. If there are no formatting commands in the text string, the encode attribute has no effect.

Example:

```
text="<h1><i>Important</i></h1>"
```

encode = "false"

Browser output:

Important

Uniform Resource Name (URN)

the text string is rendered by interpreting the formatting commands.

encode = "true" Browser output: `<h1><i>Important</i></h1>`

the formatting commands are displayed and not interpreted.

- **hasDesignBar**
Enables or disables a design bar around the label.
- **id**
Identification name of the label.
- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).
A boolean value that defines if a JavaScript object has to be generated for the inputField component.
- **labelFor**
Identification name of the next component, which is associated with the label.
- **required**
A boolean value. If set to "true" a character (for example, an asterisk (*) in red color) defined by the style sheet is placed at the end of the text string. This is a common method to indicate that input is required. See also inputField.
- **text**
Defines the string of text displayed. See 'encode' for a formatting example with embedded HTML commands.
- **tooltip**
Defines the hint of the label which is displayed as the mouse cursor passes over the label, or as the mouse button is pressed but not released.
- **valid**
Defines the status of the associated input field.
- **width**
Defines the width of the label. The width shows only effect when the 'wrapping' attribute is set to "true". Otherwise the width and layout follows the HTML commands in the text string.
- **wrapping**
A boolean value. If set to "true" the text is word wrapped at the set 'width' or - if no 'width' is set - at the form width.

Attributes	M	Values	Usage
design		LABEL LABELSMALL	<i>Taglib</i> design="LABEL" <i>Classlib</i> setDesign (TextViewDesign.LABEL)
enabled		FALSE TRUE (<i>d</i>)	<i>Taglib</i> enabled="FALSE"

Uniform Resource Name (URN)

			<i>Classlib</i> setEnabled(false)
encode		FALSE TRUE (d)	<i>Taglib</i> encode="FALSE" <i>Classlib</i> setEncode(false)
hasDesignBar		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setHasDesignBar(false)
id	*	String (cs)	<i>Taglib</i> id="Intro_text" <i>Classlib</i> setId("Intro_text")
jsObjectNeeded**		FALSE (d) TRUE	<i>Taglib</i> jsObjectNeeded="TRUE" <i>Classlib</i> setJsObjectNeeded(true)
labelFor	*	String (cs)	<i>Taglib</i> labelFor="id_inputField" <i>Classlib</i> setLabelFor("id_inputField")
required		FALSE (d) TRUE	<i>Taglib</i> required="TRUE" <i>Classlib</i> setRequired(true)
text		String	<i>Taglib</i> text="Your name please" <i>Classlib</i> setText("Your name please")
tooltip		String	<i>Taglib</i> tooltip="Name required" <i>Classlib</i> setTooltip("Name required")
valid		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setValid(false)
width		Unit (100%)	<i>Taglib</i> width="300" <i>Classlib</i> setWidth("300")
wrapping		FALSE (d) TRUE	<i>Taglib</i> wrapping="TRUE" <i>Classlib</i>

Uniform Resource Name (URN)

		setWrapping(true)
--	--	-------------------

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component.

** Method is inherited from the [Component \[Page 33\]](#) component.

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Example

using the taglib

```
<hbj:label
  id="label_InputName"
  required="TRUE"
  text="ZIP Code"
  design="LABEL"
  labelFor="InputName"
/>

<hbj:inputField
  id="InputName"
  type="string"
  maxlength="100"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
InputField input = new InputField("InputName");
input.setType(DataType.STRING);
input.setMaxLength(100);
Label label = new Label("label_InputName");
label.setRequired(true);
label.setText("ZIP Code");
label.setDesign(TextViewDesign.LABEL);
label.setLabelFor(input);
form.addComponent(label);
form.addComponent(input);
```

Result

ZIP Code *

3.4.2.6.18.1 Usage & Type

Labels are texts that describe associated input elements. The label control creates a firm connection between the descriptive text and the respective element.

Uniform Resource Name (URN)

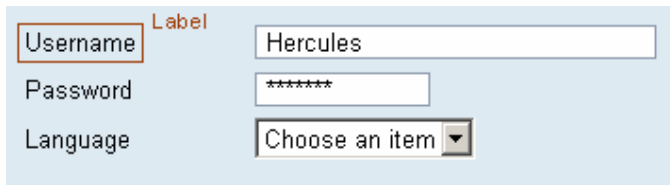


Figure 1: Example of a label with its corresponding input element

Usage

Labels are simple text elements that can be used for descriptive texts. Labels can wrap and spread over multiple lines. Use labels to describe input elements, such as, [dropdown list boxes \[Page 33\]](#), [input fields \[Page 33\]](#), [list boxes \[Page 33\]](#), and [item lists \[Page 33\]](#). [Checkbox \[Page 33\]](#) and [radio button \[Page 33\]](#) controls already include their own labels.

Required Fields

Set the label to required if the input field is required.

Positioning

Typically, labels are placed in front of the input element they describe (figure 1). In some cases, labels may be placed above the corresponding input element (figure 2). See also the special case for input fields below (figure 3).

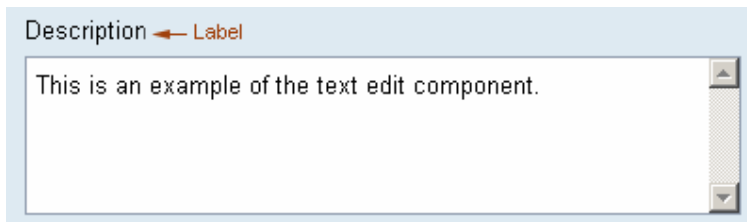


Figure 2: Label above a text edit control

Size

The label control can be set to two sizes, standard and small. Use the sizes depending on the context of the label, that is, use the standard size if the surrounding fields are standard size, use small if the surrounding fields are small.

There is one exception to this rule: You may use small labels if you place the labels above input fields to achieve a more compact design (figure 3).

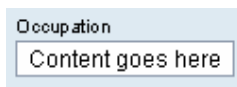


Figure 3: Use small labels for labels that are placed above their associated input fields

Relation to Text View

Do not use the [text view \[Page 33\]](#) control as a label because only the label establishes a connection between the description and the corresponding input element. This connection is mandatory for achieving accessibility.

Help Texts

Help texts are special descriptions that are placed behind or, if space is limited, below and left-aligned with input fields (figure 4). Do not use the label control for help texts. Use the [text view \[Page 33\]](#) control, instead. Set the help text size to small (style Legend).

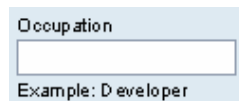


Figure 4: A small label used in conjunction with a small help text below the associated input fields - do not use the label control for the help text (below the input field)

Design-relevant Attributes

The appearance of the label control can be determined through several attributes. The attribute design allows to set the size of the label: design = LABEL sets the standard size, design = LABELSMALL sets the small size. The attribute labelFor established the connection to the associated input control. Further attributes set the width, wrapping behavior (Boolean attribute wrapping), text, and tooltip text of the label. The Boolean attribute required must be set to TRUE for labels that describe required fields

Related Controls

[Dropdown List Box \[Page 33\]](#), [List Box \[Page 33\]](#), [Input Field \[Page 33\]](#), [Text Edit \[Page 33\]](#), [Text View \[Page 33\]](#), [Item List \[Page 33\]](#)

3.4.2.6.18.2 Browser Support & 508

The Label component renders in every supported browser.

Editability in Style Editor

The label control is editable in the Style Editor via "Text". Common styles used for labels (changes affect other elements) are:

- Font Size
- Font Color
- Font Family
- Font Weight
- Font Style

Uniform Resource Name (URN)

For an overview of all common styles see section HTMLB Controls and Style Editor in Customer Branding and Style Editor.

Accessibility - 508 Support

- **Keyboard**
The label is not inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Typically, an additional description is not needed. Set a description using the `setTooltip` method only if the label text is not sufficient for explaining the meaning, function, or purpose of the corresponding control.
- **Corresponding Input Elements**
Checkbox, dropdown list box, input field, radio button, and text edit control.

3.4.2.6.19 Link

Definition

Defines a link to another page. The text of the link becomes an underline and is displayed in a different color. An image can be defined as link as well - see the example for details. The link control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).
A boolean value that enables (=true) or disables (=false) the link control. A disabled link has a different text color to show the user that it is disabled and sends no event when clicked.
- **fontSize**
Sets the font size for the link.
- **id**
Identification name of the link.
- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).
A boolean value that defines if a JavaScript object has to be generated for the link component.
- **labeled**
Enables or disables the notification when the control has a label assigned to it.
- **linkDesign**
Design definition of the link.
 - DRAGRELATE
In link color and underlined. On "MouseOver" color different, no underline

Uniform Resource Name (URN)

- **DRILLDOWN**
In link color and underlined. On "MouseOver" link color brighter and underlined. "Standard" link display.
- **FUNCTION**
In link color and underlined. On "MouseOver" link color brighter and underlined. "Standard" link display.
- **REPORTING**
In link color, no underline. On "MouseOver" link color brighter and underlined.
- **RESULT**
In text color, no underline. On "MouseOver" link color brighter and underlined.
- **onClick**
Defines the event handling method that will be processed when the user clicks on the link. If 'onClick' is specified, the event handling routine is called.
- **onClientClick**
Defines the JavaScript fragment that is executed when the user clicks on the link. If both events ('onClick' and 'onClientClick') are specified, the 'onClientClick' event handling method is activated first. By default the 'onClick' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onClick' event handling method with the command

```
htmlbevent.cancelSubmit=true;
```

The 'onClientClick' event is useful to preprocess the form and only send the form to client if the preprocessing was successful (for example, date validation, valid number format etc.) to save client/server interaction.
- **reference**
Specifies the address of the page/document to be opened. If the 'onClick' attribute is defined the event handling routine is started and the 'reference' string is handed to the event handling routine. The referenced document is not opened - the event handling routine has to do that.

If the 'onClick' attribute is not defined, the link is opening the referenced document.
- **target**
Specifies the name of the frame where the document is to be opened. The following values refer to w3c HTML-standard.
 - **_blank**
The web client should load the designated document in a new, unnamed window.
 - **_self**
The web client should load the document in the same frame as the element that refers to the target.
 - **_parent**
The web client should load the document into the immediate FRAMESET parent of the current frame. This value is equivalent to **_self** if the current frame has no parent.
 - **_top**

Uniform Resource Name (URN)

The web client should load the document into the full, original window (thus canceling all other frames). This value is equivalent to `_self` if the current frame has no parent.

- **text**

A text string that is displayed underlined and in different color. If no 'text' attribute is provided or a the 'text' attribute is set to an empty string, the link is not displayed.

- **tooltip**

Defines the hint of the link which is displayed as the mouse cursor passes over the link, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
enabled*		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEnabled (true)
fontSize			<i>Taglib</i> No tag available <i>Classlib</i> setFontSize (LinkFontSize.fontSize)
id	*	String (cs)	<i>Taglib</i> id="importantItems" <i>Classlib</i> setId("importantItems")
jsObjectNeeded**		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setJsObjectNeeded (true)
labelled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setLabelled (true)
linkDesign		DRAGRELATE DRILLDOWN (d) FUNCTION REPORTING RESULT	<i>Taglib</i> linkDesign="RESULT" <i>Classlib</i> setLinkDesign (LinkDesign.RESULT)
reference		String	<i>Taglib</i> reference="http://www.sap.com" <i>Classlib</i> setReference ("http://www.sap.com")
target		_blank _self (d) _parent _top	<i>Taglib</i> target="_TOP" <i>Classlib</i> setTarget ("_TOP")
text		String	<i>Taglib</i>

Uniform Resource Name (URN)

			<code>text="To the beach"</code> <i>Classlib</i> <code>setText("To the beach")</code>
tooltip		String	<i>Taglib</i> <code>tooltip="Enjoy and relax"</code> <i>Classlib</i> <code>setTooltip("Enjoy and relax")</code>

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the [Component \[Page 33\]](#) component.

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Events	M	Values	Usage
onClick		String (cs)	<i>Taglib</i> <code>onClick="ProcessLink"</code> <i>Classlib</i> <code>setOnClick("ProcessLink")</code>
onClientClick		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setOnClientClick("alert('Click')")</code>

Example

Text as link using the taglib

```
<hbj:link
  id="link1"
  text="Link to google"
  reference="http://www.google.com"
  target="_TOP"
  tooltip="this takes you to: http://www.google.com"
  onClick="LinkClick"
/>
```

Text as link using the classlib

```
Form form = (Form) this.getForm();
Link link = new Link("myLink");
link.setTarget("_TOP");
link.setReference("http://www.google.com");
link.setTooltip("this takes you to: http://www.google.com");
link.addText("Link to google");
form.addComponent(link);
```

Uniform Resource Name (URN)

Result[Link to google](#)

Image as link (and getting the resource path once - at the beginning of the JSP. This is useful when the JSP uses several images) using the taglib

```
<!-- Get resource url of component      --%>
<%
    String  ImageURL=componentRequest.getPublicResourcePath()
        +    "../mimes/";
    %>
<hbj:link
    id="link1"
    text=""
    reference="http://www.sap.com"
    target="_TOP"
    tooltip="this takes you to: http://www.sap.com"
    onClick="LinkClick"
    <hbj:image
        src="<%= ImageURL+"sap.gif\" %>"
        alt="Image not available"  />
    </hbj:link>
```

Result

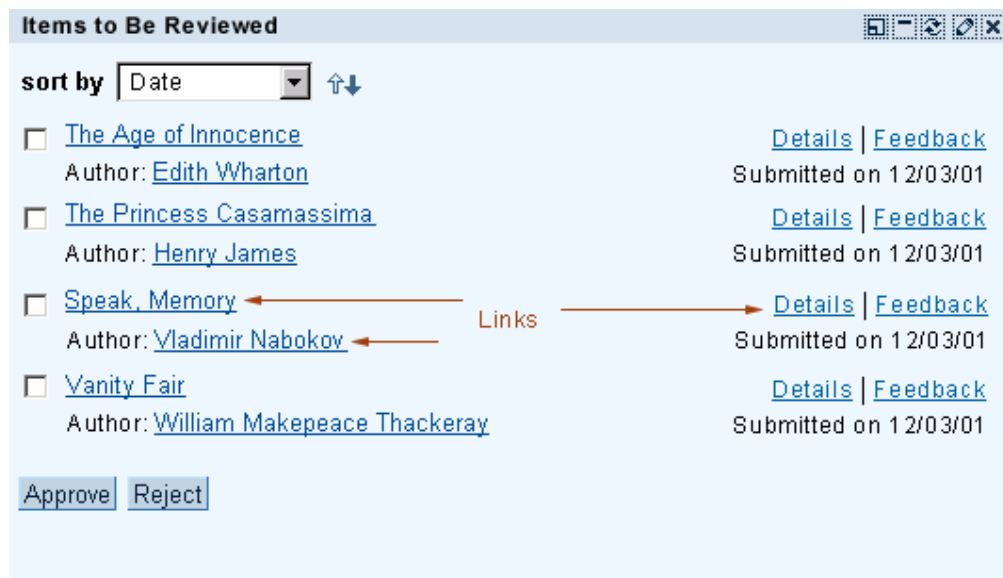
Text and Image as link (and getting the resource path at the control) using the taglib

```
<%@ page import="com.sapportals.portal.prt.resource.IResource" %>
.
.
<hbj:link
    id="link1"
    text="Link to SAP"
    reference="http://www.sap.com"
    target="_TOP"
    tooltip="this takes you to: http://www.sap.com"
    onClick="LinkClick"
    >
    <hbj:image
        id="image_logo"
        alt="Image not available"
        src=""
        >
        <%
            IResource
            rs=componentRequest.getResource(IResource.IMAGE,
                "../mimes/sap.gif");
            image_logo.setSrc
                (rs.getResourceInformation().getURL(componentRequest));
            %>
        </hbj:image>
    </hbj:link>
```

Result



3.4.2.6.19.1 Usage & Type



Fig

Figure 1: Example of links in the content area

Usage

The link control is not the simple topic it may seem to be. Not all links should be handled equally. There are hypertext links in the navigation area, header area, and content area as well as on buttons and in tables and applications. All of these links can have different appearances and behaviors.

For aesthetic and usability reasons, the links in the navigation area (not to be confused with content area navigation links!) have a different appearance from those in the content area. It's important that the user recognizes these links as pertaining to the overall navigational structure and not just a way to drill-down to detail information or jump to a page in a different context.

The HTMLB control called "link," which is described here, refers to hypertext links (referred to here as "links") in the content area.

Positioning

Links may either appear

Uniform Resource Name (URN)

- as part of a larger text, (for example, an article about customer service might contain a context link to the mySAP CRM homepage) or
- standalone (for example, a list of links to articles about CRM, a "More" link placed at the end of a abstract to navigate to additional information, etc.)

Standalone links should be grouped together separately from buttons and vice versa. When possible, functions and links should be grouped together and displayed in the same way (either all as links or all as buttons), as in figure 1. A mixture of links and buttons in the same grouping context should be avoided.

View switching links are most often displayed above the content to which they refer, as in figure 2.

Capitalization

It is not necessary to make any special capitalization considerations for links that are part of a larger context or that are automatically generated (such as contact names, document titles, etc.). In the same way that buttons require title case capitalization (i.e. the first word always capitalized, all significant words are capitalized, prepositions and articles are not capitalized), so do function, navigation, toggle and view switching links.

Market Analysis.pdf

Title case → [Show Contents of All Messages](#)

	Subject	Author	Date
Sentence case	▼ Your opinion on our market analysis	Mary Scott	May 25th 2001, 09:05
	A great challenge	Daniel Jackson	May 25th 2001, 09:09
	I agree	Charles Brown	May 25th 2001, 09:15
	Let's go for it	Mary Scott	May 25th 2001, 09:24
	Consider the risks	Daniel Jackson	May 25th 2001, 09:12
	My opinion	Charles Brown	May 25th 2001, 09:11
	▼ The numbers look suspicious	Jena Watson	May 25th 2001, 09:11
	Accounting says they'll pass	Albert Finkel	May 25th 2001, 09:11

[Create New Topic](#)
[Subscribe to this Discussion](#)

Figure 2: Example of an iView where links have both title and sentence case depending on the usage

Types

Although there is only one link type from a technical point of view, we must make a distinction between the various kinds of links in the content area. This helps to establish usage rules for links and to make a distinction between buttons and links.

Note: Based on what purpose the links serve, we can establish five different types of content area links: view switch, toggle, drill-down, function, and navigation.

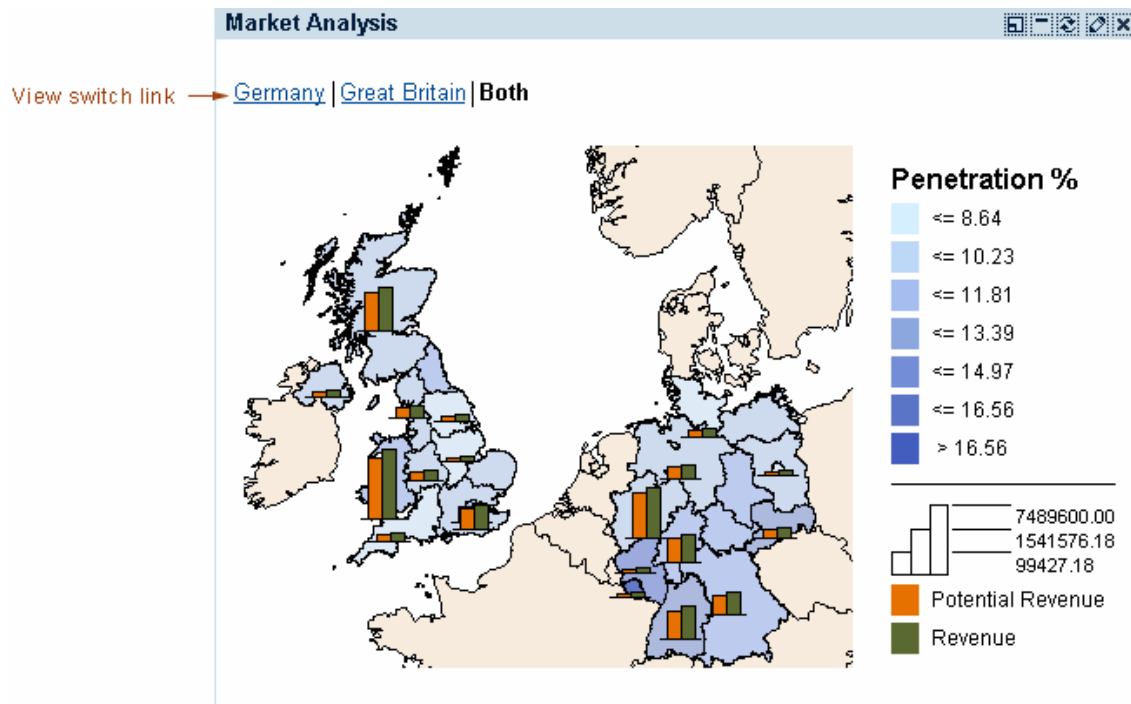


Figure 3: Example of an iView with view switch links.

View Switch Links

View switch links are similar in function to toggle links, but are different in that they are always visible. The view switch links are an alternative to the tabstrip control. The advantage that view switch links have over the tabstrip is that they take up less space and can be used vertically, if this makes more sense in the application (for example in mobile applications where the format of the device allows more vertical than horizontal space). This type of link is similar to the navigation link. As opposed to navigation links, view switch links all refer to the same context and must be persistent in all the views. The currently selected view should not be presented as a link, but as bold text (see figure 3 above). View switch links should be separated from one another by a vertical line, like this one |.

Use title case (see Capitalization) for view switch links.

Uniform Resource Name (URN)

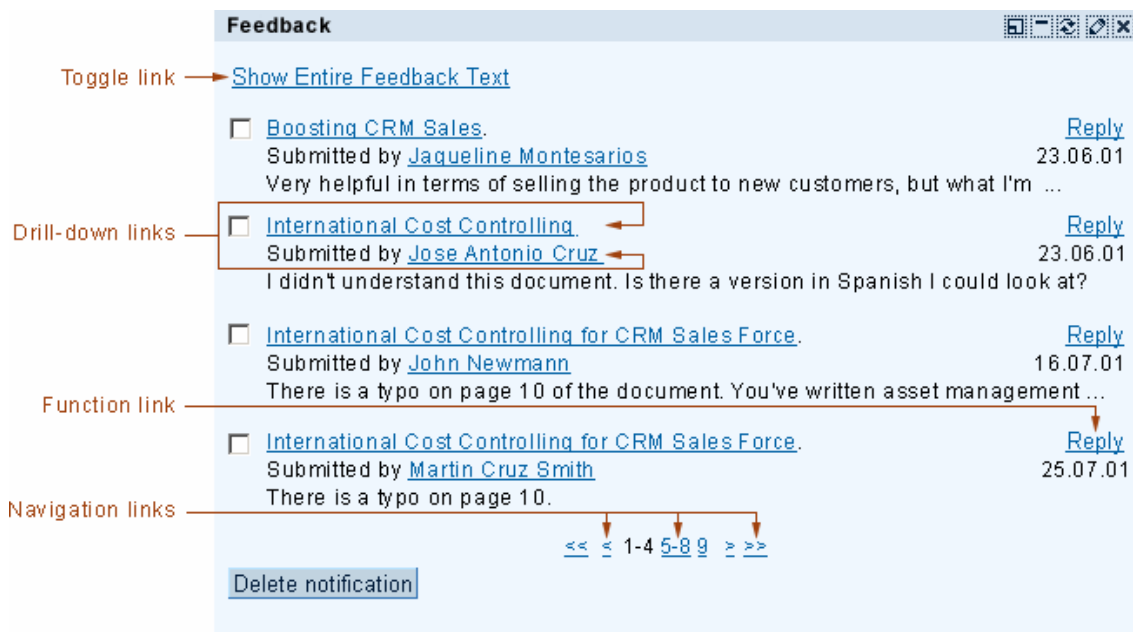


Figure 4: Example of an iView with toggle, drill-down, function and navigation links.

Toggle Links

Toggle links are used when there are two alternative views of the current data. Toggle links are always pairs of links, but only one is visible at a time. In figure 4 the toggle link reads, "Show Entire Feedback Text." If the user were to click on that link, the alternative view would show the feedback in its entirety and the text of the link would change to "Show Only Feedback Preview."

Some common examples of toggle links are:

- Expand All / Collapse All.
- Show Chart View / Show Table View.
- Hide Help / Show Help.

Use title case (see Capitalization below) for toggle links.

Drill-Down Links

A drill-down link allows a user to see more detailed or specific information. For example, in the overview of an address book, the contact names are drill-down links that allow the user to access details on that person. In a mail inbox, the title of each message would automatically be a link to the contents of the message.

Some common drill-down links are:

- Contact name
- Customer name
- Document title
- Message title

Uniform Resource Name (URN)

- Report title
- Revenue

These links are most often automatically generated and the developer should make no attempt to influence the capitalization. (See figure 2.)

Function Links

A function link allows the user to carry out an action. Although in general buttons should be reserved for functions (see Links vs. Buttons for more information), there may be cases where a link would be preferable or where the distinction between a function and a link to another view might be very blurry.



The user wants to subscribe to an object (for example, a folder), the link may just be a link to a new screen where the user has to fill in some more information and then can submit the information to the server. This whole act of subscribing, however, could be thought of as carrying out a function and in some cases might make more sense as a button than as a link, or vice versa depending on the context.

Sometimes you may have a collection of items, for example a list of documents. There are some actions that you can perform all at once on a number of documents whereas there are other actions which make more sense when they are performed on each item one at a time. If the action requires a new screen or additional information (details, feedback, edit, reply, etc.), chances are that you can only perform the action on each item one at a time. In such cases, a link is generally preferable to a button.



Taking the iView in figure 1 as an example, we can see that the user can select a number of documents and approve or reject them all at once by using the checkboxes and buttons. However, if the user were to want to see the details of a number of documents, it makes more sense to have him choose a link next to each document. Otherwise, the user would have a number of detail screens and would likely be confused about which details belong to which document.

If your application has a list of items, and each one requires its own function (see figures 1 and 4 for examples), it is preferable to use links as opposed to buttons for functions in this case. This is mainly due to aesthetics, but it is also a usability factor. An application with a wall of buttons makes the application look heavy and complicated.

Some common function links are:

- Details
- Feedback
- Add
- Subscribe
- Reply
- Edit

Use title case (see Capitalization) for function links.

Uniform Resource Name (URN)

Navigation Links

Many times, there will be navigation within an iView or application which is independent of the main navigation of the Portal. End users might not even perceive these links as "navigation" per se. Navigation links allow users to move backwards or forwards through a data set or process. Sometimes the difference between a drill-down link and a navigation link might be difficult to assess, for example with a "more..." link.

Some common navigation links are:

- More...
- Next or Next >
- Back or < Back
- Backward "<" and forward ">" as well as back to the beginning "<<" and forward to the end ">>" arrows in text form (as in figure 4 above)
- Numbers to navigate to a set of entries (as in figure 4 above)

Use title case (see Capitalization) for navigation links.

Links vs. Buttons

In general, use links to indicate navigation to another HTML page or to a different view of the current information as well as to link to further or more detailed information. Links commonly appear within the context of the application (within trees, tables or text).

In general, use buttons to indicate that a function can be carried out (save, print, close, delete, etc.) or that a process can be started (subscribe, etc.). Buttons generally appear at the bottom left of a grouped area to indicate a function that can either be performed on selected items (if checkboxes appear as well) or that apply to the whole screen. (See the section of these guidelines called Buttons for further information.)

For more information about cases where links may be used to indicate a function, see function links above.



Figure 5: Example of an iView where links and buttons are both used.

States

The link control has four states: link (i.e. unvisited, normal state), hover, visited and active.

This is the link control	Example of a link in its normal state
This is the link control	Example of a link in the hover state
This is the link control	Example of a link in the visited state
This is the link control	Example of a link in the active state

Table 1: Examples of the different states the link control can have

Design-relevant Attributes

Attribute reference allows to assign an URL to the link, whereas attribute target allows to set a link target. The latter follows the HTML conventions for specifying link targets. Attribute text sets the link text, and attribute tooltip the tooltip text for the link.

For details see page [Control API \[Page 33\]](#) for Link.

Related Controls

[Button \[Page 33\]](#)

3.4.2.6.19.2 Browser Support & 508

Netscape does not recognize the hover state.

Editability in Style Editor

In the Style Editor it is possible to edit the following styles:

- Font Color for Unvisited ("link"), Active ("active"), Visited ("visited") and Mouseover ("hover").
- Text Decoration for Unvisited ("link"), Active ("active"), Visited ("visited") and Mouseover ("hover")

For an overview of all common styles see section HTMLB Controls and Style Editor in Customer Branding and Style Editor.

Accessibility - 508 Support

- Keyboard
The link is inserted into the accessibility hierarchy by default.
- Default Description
Is provided by the HTMLB rendering engine.

Uniform Resource Name (URN)

- **Application-specific Description**

Set an additional description using the `setTooltip` method if needed.

An additional description is needed if users need more specific information or instructions. In general, the description has to be extended if a link introduces an interaction that cannot be recognized by a blind user. For example, the descriptions needs to be extended if the link opens a new window.

3.4.2.6.20 List Box

Definition

A set of choices from which a user can select one items. If the number of text lines exceeds the control size, a vertical scrollbar is activated. An item in the listBox is called listBoxItem. listBoxItems are explained above - after the dropdownListBox description. The listBox control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).

A boolean value that defines if the listBox is click able. If the listBox is disabled (enabled = false) it is not selectable. A disabled listBox has a different color for the displayed listBoxItem.

- **id**

Identification name of the listBox.

- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the listBox component.

- **labeled**

Enables or disables the notification when the control has a label assigned to it.

- **model**

Defines the model which provides the listBox with data. How to work with the [IListModel \[Page 33\]](#).

- **nameOfKeyColumn**

Specifies the name of the column that contains the key. This is used when you use an underlying table in the model.

- **nameOfValueColumn**

Specifies the name of the column that contains the visible text. This is used when you use an underlying table in the model.

- **onClientSelect**

Defines the JavaScript fragment that is executed when the user clicks on the dropdownListbox. If both events ('onSelect' and 'onClientSelect') are specified, the 'onClientSelect' event handling method is activated first. By default the 'onSelect' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onSelect' event handling method with the command

```
htmlbevent.cancelSubmit=true;
```

Uniform Resource Name (URN)

The 'onClientSelect' event is useful to pre process the form and only send the form to client if the preprocessing was successful (for example, date validation, valid number format etc.) to save client/server interaction.



A listBox click usually activates the client/server interaction. If an input field has to be filled out for further processing, the JavaScript fragment can check the necessary input on the client side and display a message if the necessary input is missing, without server interaction.



To use JavaScript the JSP has to use the page tag (see [page \[Page 33\]](#) tag).

- **onSelect**

Defines the event handling method that will be processed when the user clicks on the enabled listBox. If you do not define a 'onSelect' event the listBox can be clicked but no event is generated.

- **selection**

Specifies the key of the listBoxItem which is displayed in the listBox.

- **size**

Sets the number of lines displayed for the listBox. If the number of text lines for listBox is higher then the size attribute a vertical scrollbar is activated - the width of the listBox is not changed, the text display window becomes smaller.

- **tooltip**

Defines the hint of the listBox which is displayed as the mouse cursor passes over the listBox, or as the mouse button is pressed but not released.

- **width**

Defines the width of the listBox. Text lines are truncated if the length of the string extends the width.

Attributes	M	Values	Usage
enabled		FALSE TRUE (d)	<i>Taglib</i> disabled="TRUE" <i>Classlib</i> setEnabled(false)
id	*	String (cs)	<i>Taglib</i> id="listbox_te" <i>Classlib</i> setId("listbox_te")
jsObjectNeeded		FALSE (d) TRUE	<i>Taglib</i> jsObjectNeeded="TRUE" <i>Classlib</i> setJsObjectNeeded(true)
labeled		FALSE (d) TRUE	<i>Taglib</i> jsObjectNeeded="TRUE"

Uniform Resource Name (URN)

			<i>Classlib</i> setJsObjectNeeded(true)
model		String (cs)	<i>Taglib</i> model="myBean.model [Page 33]" <i>Classlib</i> setModel((ListModel [Page 33]) model)
nameOfKeyColumn		String (cs)	<i>Taglib</i> nameOfKeyColumn="k1" <i>Classlib</i> setNameOfKeyColumn("k1")
nameOfValueColumn		String (cs)	<i>Taglib</i> nameOfValueColumn="v1" <i>Classlib</i> setNameOfValueColumn("v1")
selection		String (cs)	<i>Taglib</i> selection="HD" <i>Classlib</i> setSelection("HD")
size		Numeric (4)	<i>Taglib</i> size="10" <i>Classlib</i> setSize("10")
tooltip		String	<i>Taglib</i> tooltip="select an item" <i>Classlib</i> setTooltip("select an item")
width		Unit (max. item length)	<i>Taglib</i> width="100" <i>Classlib</i> setWidth("100")

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the [Component \[Page 33\]](#) component.

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Events	M	Values	Usage
onClientSelect		String (cs)	<i>Taglib</i> onClientSelect="alert('Click') " <i>Classlib</i> setOnClientSelect("alert('Click') ")
onSelect [Page 33]		String (cs)	<i>Taglib</i> onSelect="proc_listbox"

Uniform Resource Name (URN)

			<i>Classlib</i> setOnSelect("proc_listbox")
--	--	--	--

To define items in the list box without a model, see [listBoxItem \[Page 33\]](#).

Example

using the taglib

```
<hbj:listBox
  id="LB_CitiesNearby"
  tooltip="Cities surrounding SAP"
  selection="WD"
  disabled="false"
  nameOfKeyColumn="KeyCol"
  nameOfValueColumn="KeyVal"
  onSelect="ProcessCity"
  onClientSelect="PreprocessCity"
>
<hbj:listBoxItem
  key="HD"
  value="Heidelberg"
  selected="true"
/>
<hbj:listBoxItem
  key="HK"
  value="Hockenheim"
/>
<hbj:listBoxItem
  key="WD"
  value="Walldorf"
  selected="true"
/>
<hbj:listBoxItem
  key="WL"
  value="Wiesloch"
/>
</hbj:listBox>
```

using the classlib

```
Form form = (Form) this.getForm();
ListBox lb = new ListBox("LB_CitiesNearby");
lb.setTooltip("Cities surrounding SAP");
lb.setWidth("300");
lb.addItem("HD", "Heidelberg");
lb.addItem("HK", "Hockenheim");
lb.addItem("WD", "Walldorf");
lb.addItem("WL", "Wiesloch");
lb.setOnSelect("ProcessCity");
lb.setOnClientSelect("PreprocessCity");
lb.setTooltip("Cities nearby");
lb.addSelection("WD");
lb.addSelection("HD");
form.addComponent(lb);
```

Uniform Resource Name (URN)

Result**3.4.2.6.20.1 Usage & Type**

The list box is a box that displays a list of items where users can select one item from. If the number of items exceeds the box size, a vertical scrollbar is activated. The list box is read-only.

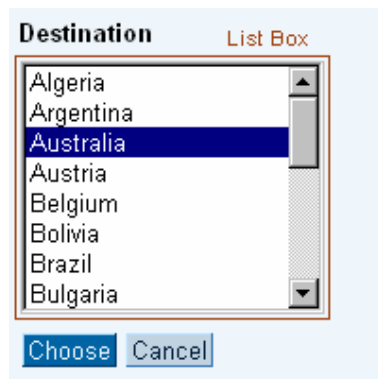


Figure 1: Example of the list box control.

Usage

A list box offers a set of choices from which a user can select one item. If the number of items exceeds the control size, a vertical scrollbar is activated. An item in the list box is called list box item. The list box is read-only.

Note: The list box control does not render a descriptive label automatically. Use the [label \[Page 33\]](#) control to add a description. See there, how you can change text attributes if you need to highlight the label, for example, make it bold (see figure 1).

Choosing the Appropriate Selection Control

A list box is similar in function to a dropdown list box - both offer a list of items where users can select one item from, that is, both are single-selection lists.

Note: For very small item numbers (2-6) and if the users should see all alternatives, use [radio buttons \[Page 33\]](#).

Design Options for Lists

HTMLB offers several controls for displaying and editing data sets and for selecting from data sets.

- Checkbox: multiple selection from small data sets (static).
- Radio Button: single selection from small data sets (static).
- List Box: Single selection from small data sets (static).
- List Box: Single selection from small to medium data sets (dynamic or static)
- Item List: Display of small to medium data sets in one column - ordered or unordered (static)
- Table View: Display and editing of data sets in a variety of display variants - in one or more columns (dynamic), single- or multiple-selection possible

Note: The cases of checkbox and radio button groups are not discussed here; see Forms - Using Checkboxes and Forms - Using Radio Buttons for details on the layout options for these controls.

Dropdown List Box vs. List Box

The overview above showed that a list box is similar in function to a dropdown list box - both offer a list of items where users can select one item from, that is, both are single-selection lists. Here you find criteria for choosing between both controls. In addition, we provide hints when a set of radio buttons is the appropriate choice.



Figure 1: Dropdown list box (left) vs. list box

When Use a List Box?

- If there is more space on the page.
- For larger item numbers.
- If users need to know the context of the current selection, that is, the item set or at least part of it.
- If users need to carefully consider their choice.
- For users with low mouse abilities.

When Use a Dropdown List Box?

- In table views.
- If space is limited - it occupies one line only.
- For smaller item numbers (up to 20 items).
- If users only need to know the current selection, not the whole set.
- In shufflers

When Use Radio Buttons?

For very small item numbers (2-6) and if the users should immediately see all available alternatives, use radio buttons.

Use larger radio button sets only in special cases, where it is important that all options are visible, where users are untrained, and/or where an application imitates a paper-form, such as a Web-based questionnaire or ordering form.

Note: For multiple choices use checkboxes instead of radio buttons.

Item List vs. List Box

Both item lists and list boxes can be used for displaying a set of options. While list boxes can also be used for selecting items we focus here on the display aspect. Item lists are static lists with a "paper-like" appearance; they can be ordered or unordered. List boxes, however, have a "form-like" appearance and also may contain more items than are visible.

If you consider to use a list box, you may check whether a table might also be a valid design option. We also provide some hints for this option.



Figure 2: Item list (ordered, left) vs. list box

When Use an Item List?

- In paper-like applications, such as news, articles, etc.

Uniform Resource Name (URN)

- If it is possible to display all items or if the page or application as a whole can be scrolled.
- If the number of items is fixed

When Use a List Box?

- In form-like applications, typically together with other form elements, such as input fields and selection elements.
- If a selection is needed.
- If space may not suffice to display the whole list.
- If the application or page cannot be scrolled in order to display more items.
- If bullets or numbers should not appear.

When May a Table View Be Used?

In the following, we list scenarios, where a table view may be used instead of a list box. Note that this is an option to consider, not a recommendation. The only exception is multiple-selection, which is available for the table view only.

Typically, you would use the table view in the transparent design for this application. Also note that a table view introduces "visual overhead", such as the title, column headers, and scroll buttons.

- In form-like applications where the data form a separate information unit that cannot be mixed with fields.
- Where a multiple-column display is needed.
- Where scroll buttons are preferred over scrollbars (page-wise scrolling).
- Where multiple-selection is needed (this is currently not possible with list boxes)

Note: Static multiple-selection can also be implemented using a checkbox group.

Design-Relevant Attributes

You can set the number of displayed lines of a list box (size), its width (width), and whether it is enabled or disabled (Boolean attribute disabled).

See Control API for [List Box \[Page 33\]](#) for details.

Related Controls

[Dropdown List Box \[Page 33\]](#), [Item List \[Page 33\]](#), [Radio Button \[Page 33\]](#), [Table View \[Page 33\]](#), [Tree View \[Page 33\]](#)

3.4.2.6.20.2 Browser Support & 508

Uniform Resource Name (URN)

Renders in every supported browser.

Editability in Style Editor

The list box itself renders as the standard browser control. Style Editor changes can be made to the corresponding label.

Accessibility - 508 Support

List boxes have to be used in combination with the label element which points to the assigned list box. This ensures, that screenreaders are aware of the relationship between the both elements and can read the correct label to the according list box.

- **Keyboard**
The listbox inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed.
- **Label**
Has to be connected to a label control (use method `setLabelFor` for identifying the corresponding list box).

3.4.2.6.21 List Box Item

Purpose

Defines the items in a `dropdownListBox` or `listBox` instead of the model.

- **key**
A string which is passed on to the event handling routine when the event occurs. A key string must be defined and must not be empty. Every `listBoxItem` must have a unique key.
- **selected**
A boolean value.
 - `selected="false"`: no effect
 - `selected="true"`
`dropdownListBox`:
The item is displayed in the `dropdownListBox`. It overrules the "selection" attribute of the `dropdownListBox`. If several `listBoxItems` are selected the last defined `listBoxItem` is displayed in the `dropdownListBox`.
`listBox`:
`selected="true"`: The item is displayed as selected in the `listBox`.

Uniform Resource Name (URN)

- **value**

Defines the text string displayed in the dropdownListBox or listBox. A 'value' string has be defined and must not be empty.

Attributes	M	Values	Usage
key	*	String (cs)	<i>Taglib</i> key="WD" <i>Classlib</i> addItem ("WD", "Walldorf")
selected		TRUE FALSE (d)	<i>Taglib</i> selected = "TRUE" <i>Classlib</i> setSelection ("WD")
value		String	<i>Taglib</i> value="Walldorf" <i>Classlib</i> see attribute "key"

Example

[Dropdown List Box \[Page 33\]](#)

[List Box \[Page 33\]](#)

3.4.2.6.22 Menu Bar

Definition

A control to create a menu bar. The menu bar is combined with a hover menu to create a pull down command structure like in a desktop application.

- **addItem**

Method to add items to the menu bar.

- **id**

Identification name of the menu bar.

- **menuTrigger**

Sets the trigger mechanism which will open the hover menu.

- ONCLICK

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the left mouse button is clicked - like in a pop up menu. This option is default for web browser Netscape 4.

- ONCONTEXTMENU

Uniform Resource Name (URN)

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the right mouse button is clicked - like for a context menu.

- **ONLRCLICK**

Hover menu is displayed when the mouse pointer is on the item linked to the hover menu and the right or left mouse button is clicked.

- **ONMOUSEOVER**

Hover menu is displayed when the mouse pointer is moved over the item linked to the hover menu. This option is default for web browsers IE5 and up and Netscape 6.

- **width**

Defines the width of the menu bar. The width of the button is automatically adjusted to the length and amount of the 'menuItem'. To see an effect of the 'width' attribute, 'width' has to be set higher as the width defined through the amount and length of the 'menuItem'.

Attributes	M	Values	Usage
addMenuItem		STANDARD (d) SMALL EMPHASIZED	<i>Taglib</i> No tag available <i>Classlib</i> addMenuItem (MenuItem.menuItem)
id	*	String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setId ("myMenuBar")
menuTrigger		ONCLICK ONCONTEXTMENU ONLRCLICK ONMOUSEOVER	<i>Taglib</i> No tag available <i>Classlib</i> setMenuTrigger (HoverMenuTrigger.ONCLICK)
width		Unit (10)	<i>Taglib</i> No tag available <i>Classlib</i> setWidth ("125px")

MenuItem

Defines the items for the menu bar.

- **enabled**

A boolean value to enable (=TRUE) or disable (=FALSE) the menu item. A disabled menu item is displayed but fires no event when the item is selected (and an event handling method has been defined for the menu item).

- **hoverMenu**

Sets the hover menu for the menu item. The hover menu is opened when the action defined by the menu trigger is performed on the menu item.

- **id**

Uniform Resource Name (URN)

Identification name of the menu item.

- **text**

A string that is displayed for the menu item. A text string must be defined and must not be empty.

See also [hoverMenu \[Page 33\]](#).

Example

using the classlib

Uniform Resource Name (URN)

```

Form form = (Form) this.getForm();
MenuBar menuBar = new MenuBar("myMenu");
menuBar.setWidth("100%");
menuBar.setDesign(MenuBarDesign.TRANSPARENT);
MenuItem item = new MenuItem("item1", "File");
HoverMenu menufile = this.buildHoverMenuFile();
item.setHoverMenu(menufile);
menuBar.addMenuItem(item);
MenuItem item2 = new MenuItem("item2", "Edit");
HoverMenu menuedit = this.buildHoverMenuEdit();
item2.setHoverMenu(menuedit);
menuBar.addMenuItem(item2);
form.addComponent(menuBar);

//      Methods to define the FILE and EDIT hover menus (sub menus)
public HoverMenu buildHoverMenuFile() {

    HoverMenu hover = new HoverMenu("hoverfile");

    HoverMenuItem file1 = hover.addMenuItem("file1", "New");
    HoverMenu menu = file1.addSubHoverMenu("filesub1");
    hover.setOnHoverMenuClick("onHoverClicked");
    menu.addMenuItem("file11", "Project");
    menu.addMenuItem("file12", "Folder");
    menu.addMenuItem("file13", "File");

    file1.setHoverItemDivider(true);
    hover.addMenuItem(new HoverMenuItem("file2", "Close"));
    hover.addMenuItem(new HoverMenuItem("file3", "Close All"));
    HoverMenuItem file4 = hover.addMenuItem("file4", "Save All");
    file4.setEnabled(false);
    file4.setHoverItemDivider(true);
    return hover;
}

public HoverMenu buildHoverMenuEdit() {
    HoverMenu hover = new HoverMenu("hoveredit");

    HoverMenuItem edit1 = hover.addMenuItem("edit1", "Undo");
    hover.addMenuItem(new HoverMenuItem("edit2", "Redo"));
    HoverMenuItem edit3 = hover.addMenuItem("edit3", "Preferences");

    edit3.setHoverItemDivider(true);
    HoverMenu editmenus3 = edit3.addSubHoverMenu("editsub3");
    HoverMenuItem edit31 = editmenus3.addMenuItem("edit31", "SAP");
    HoverMenu editmenus31 = edit31.addSubHoverMenu("editsub31");
    HoverMenuItem edit311 =
        editmenus31.addMenuItem("edit311", "Link to SAP");
    edit311.setOnItemClick("itemClicked");
    edit311.setLinkReference("http://www.sap.com");
    HoverMenuItem edit312 =
        editmenus31.addMenuItem("edit312", "Link to mySAP");
    edit312.setOnItemClick("itemClicked");
    edit312.setLinkReference("http://www.mysap.com");
    HoverMenuItem edit32 = editmenus3.addMenuItem("edit32", "About");

    return hover;
}

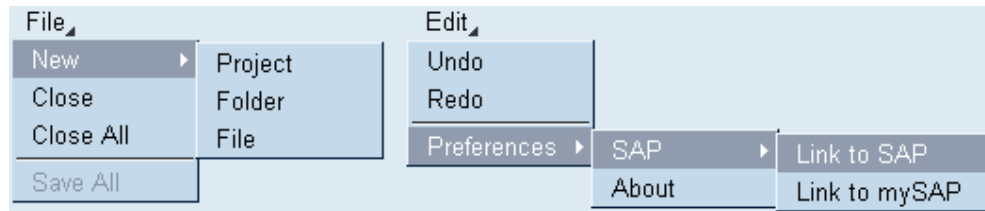
```

Result

When page is loaded:






Expanded view of all sub options:




3.4.2.6.23 Message Bar

Definition

The message bar allows to display application messages at the beginning (top) or at the end (bottom) of the form. The message can display an icon at the beginning of the messages that indicates the nature of the message (info, error and so on). The message bar has to be added to the form component. The form component has the methods to define if the message bar is displayed at the beginning or at the end of the form.

- **id**
Identification name of the message bar.
- **message**
Defines the messageType and the messageText for the message bar.
- **messageText**
Defines the messageText for the message bar. A messageType has to be defined, using method 'message' or 'messageType'. If no messageType is defined, the messageText is not displayed.
- **messageType**
Nature of the message.
 - ERROR
Displays icon .
 - INFO
Displays icon .
 - LOADING
Displays the rotating icon .
 - NONE

Uniform Resource Name (URN)

- Displays no icon.
- STOP
 - Displays the ERROR icon.
- VALIDATION
 - Displays ERROR icon .
- WARNING
 - Displays icon .

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="mBar" <i>Classlib</i> Defined with the messageBar object.
message		String	<i>Taglib</i> No tag available <i>Classlib</i> setMessage(MessageType.type, String mess)
messageText		String	<i>Taglib</i> messageText="Invalid date" <i>Classlib</i> setMessageText("Invalid date")
messageType		ERROR INFO LOADING NONE STOP VALIDATION WARNING	<i>Taglib</i> setMessage(MessageType.type, String mess) <i>Classlib</i> setMessage(MessageType.type, String mess)

Example

using the taglib

Uniform Resource Name (URN)

```
<hbj:form
  id="myFormId"
  method="post"
  target="_blank"
  encodingType="multipart/form-data"
  action="/htmlb/servlet/com.sap.htmlb.test.MyTestJsp1Test"
>
<hbj:messageBar
  id="mBar"
  messageType="ERROR"
  messageText="Fatal Error">
  <%
    myFormId.setMessageBar(mBar);
  %>
</hbj:messageBar>
</hbj:form>
```


using the classlib

```
Form form = (Form) this.getForm();
MessageBar mb = new MessageBar("messageBar");

mb.setMessage(MessageType.ERROR, "Fatal Error");
form.setMessageBar(mb);

form.setMessageBarAtFormEnd(true);

form.setMessageBarRequired(true);
```

Result Fatal Error**3.4.2.6.24 Nonisolated HTML Container****Definition**

The NonIsolatedHtmlContainer will display a HTML-document, represented by an inputStream, inside an HTML Business for Java application. The NonIsolatedHtmlContainer provides a parser, which adjusts the HTML-document so it can be displayed without iFrames. The application developer can also provide its own parser that is tailored for the application.



Documents which include Forms, Frames or Framesets can not be displayed in a NonIsolatedHtmlContainer control. If the document contains one of these tags, an error message will be displayed.

Limitations of the NonIsolatedHtmlContainer control:

Uniform Resource Name (URN)

- Relative URLs used in JavaScript will not be replaced by absolute URLs (like. `window.open('/test.htm')`).
- Style sheets will be loaded but should not contain relative URL's.
- Style sheets can effect the entire portal page.
- Correct JavaScript execution can not be guaranteed.
- Documents must be in HTML format.
- Incorrect formatted documents can destroy the entire portal page.
- Absolute positioning of the document (for example, with JavaScript) is not supported
- The exact width of the document can not be set.

Because of other limitations that can apply make sure that HTML document is formatted correctly and produces a correct page. If you run into one of the limitations you have to create your own parser.

- **bordered**
A boolean value that, if set to true, draws a border around the NonIsolatedHtmlContainer control.
- **id**
Identification name of the NonIsolatedHtmlContainer control.
- **htmlParser**
Sets a user defined HTML parser to parse the HtmlStream.
- **htmlStream**
Sets the HtmlStream of the NonIsolatedHtmlContainer control.
- **srcUrl**
Specifies the address of the page/document to be displayed in the NonIsolatedHtmlContainer control.
- **width**
Specifies the width of the NonIsolatedHtmlContainer control.

Attributes	M	Values	Usage
bordered		FALSE (d) TRUE	<i>Taglib</i> <code>bordered="TRUE"</code> <i>Classlib</i> <code>setBordered(true)</code>
id	*	String (cs)	<i>Taglib</i> <code>id="nonisoHTMLCon"</code> <i>Classlib</i> <code>setId ("nonisoHTMLCon")</code>
htmlParser		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setHtmlParser(HtmlParser.htmlParser)</code>
htmlStream	*	String (cs)	<i>Taglib</i>

Uniform Resource Name (URN)

			htmlStream="myStream" <i>Classlib</i> setHtmlStream(java.io.InputStream htmlst)
srcUrl	*	String (cs)	<i>Taglib</i> srcUrl="http://www.sap.com" <i>Classlib</i> setSrcUrl("http://www.sap.com")
width		Unit	<i>Taglib</i> width="500" <i>Classlib</i> setWidth("500")

Example

using the taglib

```
<hbj:nonIsolatedHtmlContainer
  id="nonIsohtmlCont"
  width="400"
  srcUrl="http://www.sap.com"
  htmlStream="myStream"
  bordered="true"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
URL u = new URL("http://www.sap.com");

in = u.openStream();
NonIsolatedHtmlContainer niso = new NonIsolatedHtmlContainer(in);

niso.setSrcURL(myUrl);

niso.setBordered(true);
form.addComponent(niso);
```

Result

SAP Web page is displayed

3.4.2.6.25 Progress Indicator

Definition

A progress indicator (or progress bar) bar indicates that one or more operations is under way and shows users what proportion of the operation has been completed. The progress indicator consists of a rectangular bar that fills as the operation progresses. Another

Uniform Resource Name (URN)

application for the progress indicator is to use it as a gauge to show the actual value, for example a temperature.

- **barColor**

Defines the color of the bar which moves inside the progress indicator. Following bar colors are available (specified color is based on the standard style sheet):

- CRITICAL
Bar in progress bar has orange color.
- NEGATIVE
Bar in progress bar has red color.
- NEUTRAL
Bar in progress bar has blue color.
- POSITIVE
Bar in progress bar has green color.

- **displayValue**

Defines the text that is displayed left aligned in the progress bar.

- **id**

Identification name of the progress indicator.

- **percentValue**

Defines the value in percent that the the bar inside the progress indicator should display. The value specified must be greater than 0. If you do not set this attribute an empty frame, with no progress bar inside, is displayed.

- **showValue**

A boolean value. If set to "true" the text defined with the 'displayValue' attribute is displayed together with the progress bar. If set to 'false' only the progress bar is displayed.

- **tooltip**

Defines the hint of the progress bar which is displayed as the mouse cursor passes over the progress bar, or as the mouse button is pressed but not released.

- **width**

Defines the width of the progress bar.

Attributes	M	Values	Usage
barColor		CRITICAL NEGATIVE NEUTRAL POSITIVE	<i>Taglib</i> barColor="POSITIVE" <i>Classlib</i> setBarColor (ProgressIndicatorBarColor.POSITIVE)
displayValue		String	<i>Taglib</i> displayValue="Positive" <i>Classlib</i> setDisplayValue ("Positive")
id	*	String (cs)	<i>Taglib</i>

Uniform Resource Name (URN)

			<code>id="progressBar"</code> <i>Classlib</i> Defined with progressBar object.
percentValue	String Numeric		<i>Taglib</i> <code>percentValue="20"</code> <i>Classlib</i> <code>setPercentValue(20)</code>
showValue	FALSE TRUE (d)		<i>Taglib</i> <code>showValue="FALSE"</code> <i>Classlib</i> <code>setShowValue(false)</code>
tooltip	String		<i>Taglib</i> <code>tooltip="pressure in valve 1"</code> <i>Classlib</i> <code>setTooltip("pressure in valve 1")</code>
width	Unit (100)		<i>Taglib</i> <code>width="125px"</code> <i>Classlib</i> <code>setWidth("125px")</code>

Example

using the taglib

Uniform Resource Name (URN)

```
<hbj:progressIndicator
  id="myProgressIndicator1"
  percentValue="75"
  displayValue="Critical"
  barColor="CRITICAL"
  width="300px"
/>
<hbj:progressIndicator
  id="myProgressIndicator2"
  percentValue="10"
  displayValue="Negative"
  barColor="NEGATIVE"
  width="300px"
/>
<hbj:progressIndicator
  id="myProgressIndicator3"
  percentValue="50"
  displayValue="Neutral"
  barColor="NEUTRAL"
  width="300px"
/>
<hbj:progressIndicator
  id="myProgressIndicator4"
  percentValue="90"
  displayValue="Positive"
  barColor="POSITIVE"
  width="300px"
/>
```

using the classlib

Uniform Resource Name (URN)

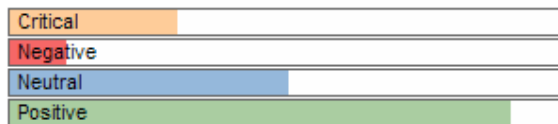
```

Form form = (Form) this.getForm();
ProgressBar pb1 = new ProgressBar();
pb1.setPercentValue("75");
pb1.setDisplayValue("Critical");
pb1.setBarColor(ProgressIndicatorBarColor.CRITICAL);
pb1.setWidth("300px");
form.addComponent(pb1);
ProgressBar pb2 = new ProgressBar();
pb2.setPercentValue("10");
pb2.setDisplayValue("Negative");
pb2.setBarColor(ProgressIndicatorBarColor.NEGATIVE);
pb2.setWidth("300px");
form.addComponent(pb2);

ProgressBar pb3 = new ProgressBar();
pb3.setPercentValue("50");
pb3.setDisplayValue("Neutral");
pb3.setBarColor(ProgressIndicatorBarColor.NEUTRAL);
pb3.setWidth("300px");
form.addComponent(pb3);

ProgressBar pb4 = new ProgressBar();
pb4.setPercentValue("90");
pb4.setDisplayValue("Positive");
pb4.setBarColor(ProgressIndicatorBarColor.POSITIVE);
pb4.setWidth("300px");
form.addComponent(pb4);

```

Result**3.4.2.6.26 Radio Button****Definition**

A button that a user clicks to set an option. Unlike checkboxes, radio buttons are mutually exclusive - selecting one radio button menu item deselects all others in that group. That is also the reason why you cannot define a radioButton by itself - it always has to be defined within a radioButtonGroup. The radioButton control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

- **enabled** - inherited from the [EventValidationComponent \[Page 33\]](#).

A boolean value that defines if the radioButton is click able. If the radioButton is disabled (enabled = false) it is not selectable. A disabled radioButton has a different background color for the radioButton graphic and if the radioButton is checked the a different color for the button mark.


- **encode**

Uniform Resource Name (URN)

A boolean value that defines how the radioButton text is interpreted. HTML text formatting commands (for example, <h1>, <i> etc.) can be used to change the display of the radioButton text. If there are no formatting commands in the radioButton text string, the encode attribute has no effect.

Example:

```
text="<h1><i>Walldorf</i></h1>"
```

encode = "false" Browser output:  **Walldorf**

the text string is rendered by interpreting the formatting commands.

encode = "true" Browser output:  <h1><i>Walldorf</i></h1>

the formatting commands are displayed and not interpreted

- **id**

Identification name of the radioButton.

- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the radioButton component.

- **key**

A string which is assigned to the radioButton when the form is sent to the server. A key string must be defined and must not be empty.

- **labeled**

Enables or disables the notification when the control has a label assigned to it.

- **selected**

Sets the status of the radio button. Is selected set to "true", a filled circle is displayed inside the radio button.

- **text**

Defines the string of text placed right of the radiobutton graphic. If no text should be displayed an empty string (null) can be used. See 'encode' for a formatting example with embedded HTML commands.

- **tooltip**

Defines the hint of the radioButton which is displayed as the mouse cursor passes over the radioButton, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
enabled*		FALSE TRUE (d)	<i>Taglib</i> disabled="TRUE" <i>Classlib</i> setEnabled (false)
encode*		FALSE TRUE (d)	<i>Taglib</i> encode="FALSE" <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setEncode(false)</code>
id	*	String	<i>Taglib</i> <code>id="GenderInfo"</code> <i>Classlib</i> <code>setId("GenderInfo")</code>
jsObjectNeeded		FALSE (d) TRUE	<i>Taglib</i> <code>jsObjectNeeded="TRUE"</code> <i>Classlib</i> <code>setJsObjectNeeded(true)</code>
key		String	<i>Taglib</i> <code>key="rb_k1"</code> <i>Classlib</i> <code>setKey("rb_k1")</code>
labeled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setLabelled(true)</code>
selected		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setSelected(true)</code>
text		String	<i>Taglib</i> <code>text="female"</code> <i>Classlib</i> <code>setText("female")</code>
tooltip		String	<i>Taglib</i> <code>tooltip="I am female"</code> <i>Classlib</i> <code>setTooltip("I am female")</code>

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component. Therefore the attribute is different between the taglib and the classlib.

** Method is inherited from the [Component \[Page 33\]](#) component.

See the [JavaScript API \[Page 33\]](#) description for details how to access the component in JavaScript.

Example

A `radioButton` has to be used together with a [radioButtonGroup \[Page 33\]](#). Refer to the [radioButtonGroup \[Page 33\]](#) documentation for the example.

3.4.2.6.26.1 Usage & Type

Radio buttons provide users with a single choice from a set of alternative options.

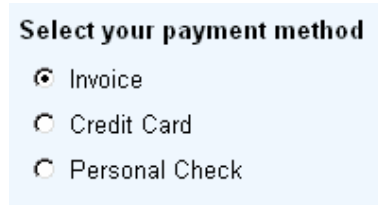


Figure 1: A radio button group

Usage

Radio buttons provide users with a single choice from a set of alternative options. They always appear in a group of at least two radio buttons. Therefore, you should define radio buttons only within the radio button group control, not as single elements.

A click on one choice selects the current choice and deselect the previous choice. Usually, always one radio button is checked. The Internet introduced one exception to this rule. In some cases, a radio button group can initially show up with no radio button checked.

Note: It is not possible to determine the horizontal spacing within a radio button group. If you need a different spacing than that supplied by the radio button group control, use single radio buttons and a [grid layout \[Page 33\]](#) control if applicable.

Arrangement and Design Alternatives

Radio button groups offer users a set of alternative choices that may be arranged either horizontally (2-3 radio buttons), vertically (not more than about 12 radio buttons), or in a matrix-like fashion. Note that radio button groups are appropriate for static and relative small numbers of options only. Use the [table view \[Page 33\]](#) for larger and dynamic data sets.

Alignment

For the alignment of radio buttons we distinguish the following cases:

- Radio buttons that refer to adjacent fields.
- Radio buttons that do not refer to elements but should be included in field groups.
- Radio buttons that can be arranged as an independent block of information

In general, the first two cases are not as common as for checkboxes.

Case 1: Radio Buttons that Refer to One or More Fields

Align dependent radio buttons with the left border of other input elements (figure 1). Place the radio button labels right to the radio button (this is done automatically for the radio button control).

Uniform Resource Name (URN)

Figure 2: A pair of radio buttons that refers to the input field above it (profession)

If there are too many alternatives, consider using a dropdown list box instead of the radio button group. As noted below, this arrangement should only be used if there is a dependency from other fields above the radio button group.

Case 2: Radio Buttons that are Included in a Field Group

If radio buttons are included in a field group but do not refer to a certain field, do the following (figure 3a-b):

- Place the description of the radio button group to the left of the group and align it with the other field labels.
- Align the radio buttons with the other input fields.
- Use a label control for the group label

You can use a vertical radio button group, or if space permits a horizontal radio button group that occupies only one line.

Figure 3a-b: Radio buttons within a field group with group label to the left (left shows vertical, right shows horizontal arrangement)

In general, radio button groups within a field group should have a descriptive label for the group and a label to the right of each radio button.

Rationale: A radio button group is functionally equivalent to a dropdown list box. The group label corresponds to the label for the dropdown list box; the labels to the right of the radio buttons correspond to the list box items

Uniform Resource Name (URN)

Figure 3c-d shows two examples of form layouts for a Uniform Resource Name (URN) field. The left example shows a form with labels 'First name', 'Last name', 'Female', 'Male', 'City', and 'Street' on the left, and radio buttons for 'Female' and 'Male' in the middle. The right example shows a form with labels 'First name', 'Last name', 'City', and 'Street' on the left, and radio buttons for 'Female' and 'Male' on the right.

Wrong level of labels (male/female is are subcategories).

Wrong alignment because the radio buttons do not refer to the name fields.

Figure 3c-d: Radio buttons within a field group with two labels to the left of the radio buttons (left) and to the right (right)

Do Not

- A layout without a label for the radio button group and with labels to the left (figure 3c) is harder to understand because the labels are not on the same semantic level as the surrounding field labels.
- Also do not use an arrangement without a group label in this case (figure 3d) because it is equally hard to understand and may lead to misinterpretations. Such a layout suggests a dependency from the field above the group to the user. Even though the layout in figure 3d is the same as in 1a, the usage is incorrect because the radio buttons do not refer to the "Last name" field. Therefore, use it only if such a dependency does exist (case 1).

Case 3: Radio Buttons that Form an Independent Information Block

If radio buttons are arranged in a separate radio button group, arrange them in a matrix-like fashion and left-align them with other elements on the page or in the application. Such groups have either to be included in a [group \[Page 33\]](#) control (see figure 4a) or separated from the field group by white space (figure 4b).

Figure 4a-b shows two examples of form layouts for a radio button group. The left example shows a form with a label 'Select your payment method' and three radio buttons: 'Invoice', 'Credit Card', and 'Personal Check'. The right example shows a form with labels 'First name', 'Last name', 'City', 'Street', and 'Gender' on the left, and radio buttons for 'Female' and 'Male' on the right.

Figure 4a-b: Radio button group that forms an information unit of its own - either included in a group (left) or separated by an empty line (right)

Uniform Resource Name (URN)

Instead of a matrix, you can use a horizontal arrangement if there are only few radio buttons. In this case, set the `columnCount` attribute of the radio button group control to a value that results in one row only.

There are two possible arrangements for horizontal radio button groups:

- The radio button row can be introduced by a label to the left or a header above - in this case align the radio buttons with other elements and use the label control for the label or header (figure 5a-b).
- The radio button row does not have an introductory label (figure 5c-d).

Separate the horizontal radio button group from preceding fields by an empty line.

The figure displays two user interface examples for horizontal radio button groups. The left example shows a form with four text input fields: 'First name', 'Last name', 'City', and 'Street'. Below these fields is a radio button group for 'Gender' with options 'Female' and 'Male'. The right example shows a form with four text input fields: 'First name', 'Last name', 'City', and 'Street'. Above these fields is a radio button group with options 'Mrs.', 'Mr.', and 'Company'.

Figure 5a-d: Horizontal radio button groups, either with an introductory label to the left (top left), a heading (bottom left), or without an introductory label (right)

Do not use a single radio button because there are two problems with it: (1) Users cannot deselect a single radio button; after it has been selected, it remains so. (2) Users see the name of one option only; they often can only guess what the alternative option is.

Dependent Fields

In some cases, the state of an input field, dropdown list box, or other control may depend on the setting of a radio button group. Below we present two simple examples, where users may either enter their nationality (figure 6a), or their payment method (figure 6b). The first radio button describes the default case; if it is set, the input fields below it are read-only. The second radio button describes the less frequent case; if it is set, the dependent input fields are ready for input. Alternatively, in the first example a dropdown list box could be used instead of the input field if the alternatives are known and their number is not too large.

Uniform Resource Name (URN)

The figure consists of two side-by-side form panels. The left panel has a light blue background and contains the following elements: 'First name' with an asterisk and a text input field; 'Last name' with an asterisk and a text input field; 'Nationality' with two radio buttons, 'American' (unselected) and 'Other' (selected); an empty text input field; 'City' with a text input field; and 'Street' with a text input field. The right panel also has a light blue background and contains: 'First name' with an asterisk and a text input field; 'Last name' with an asterisk and a text input field; 'City' with a text input field; 'Street' with a text input field; 'Payment method' with two radio buttons, 'Invoice' (unselected) and 'Credit Card' (selected); 'Card' with a dropdown menu showing 'Visa'; 'Number' with a text input field; and 'Expired' with a text input field.

Figure 6a-b: Vertical radio button group that controls the editability of one (left) or several (right) input fields below the group

Do not use this layout for more than one dependent element. If there are more dependent elements indent the dependent group so that the labels are left aligned with other input fields (figure 6b).

Design Alternatives

Radio buttons are similar in function to [dropdown list boxes \[Page 33\]](#) and [list boxes \[Page 33\]](#) with respect to offering users a single choice. Use radio buttons for very small item numbers (2-6) and if the users should immediately see all alternatives.

Design-relevant Attributes

Radio buttons have the disabled attribute. Set disabled to TRUE if users are not allowed to change their state temporarily. Attribute text sets the descriptive label text for a radio button.

For radio button groups there are two relevant attributes: You can determine which radio button is "on" in a group; set attribute selection to the id of the respective radio button. You can also set the column count for radio button groups (attribute `columnCount`).

Related Controls

[Dropdown List Box \[Page 33\]](#), [Checkbox \[Page 33\]](#), [List Box \[Page 33\]](#), [Label \[Page 33\]](#), [Grid Layout \[Page 33\]](#)

3.4.2.6.26.2 Browser Support & 508

The radio button renders in every supported browser.

Editability in Style Editor

The radio button itself renders as the standard browser control. Style Editor changes can be made to the corresponding label.

Radio Button Groups

There is no editability for radio button groups in the style editor.

Accessibility - 508 Support

If radio buttons are used with a label to the left, they have to be used in combination with the label control which points to the assigned radio button. This ensures, that screen readers are aware of the relationship between the both elements and can read the correct label to the according radio button.

- **Keyboard**
Radio buttons are inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed.
- **Label**
Has to be connected to a label control for left-hand labels (use method `setLabelFor` for identifying the corresponding radio button or radio button group).

3.4.2.6.27 Radio Button Group

Definition

Places several radiobuttons in tabular form. Only one radiobutton can be on at any given time.

- **columnCount**
Defines the amount of columns in which the radiobuttons are divided.

Example

If the `columnCount` is set to 3 and you define 7 radiobuttons the result is:

☐ Choice 1 ☒ Choice 2 ☐ Choice 3
☐ Choice 4 ☐ Choice 5 ☐ Choice 6
☐ Choice 7

- **currentItem**
Defines the item which has the focus.
- **id**
Identification name of the `radioButtonGroup`.

Uniform Resource Name (URN)

- **onClick**

Defines the event handling method that will be processed when the user clicks on one radio button. If 'onClick' is specified, the event handling routine is called.

- **selection**

Specifies the key of the radioButton that is on in the radioButtonGroup.

- **verticalAlign**

Sets the vertical alignment of the radiobutton column and the text column.

Attributes	M	Values	Usage
columncount		Numeric (1)	<i>Taglib</i> columnCount="3" <i>Classlib</i> setColumnCount(3)
currentItem		Numeric (1)	<i>Taglib</i> No tag available <i>Classlib</i> setCurrentItem(2)
id	*	String (cs)	<i>Taglib</i> id="Genderselect" <i>Classlib</i> setId("Genderselect")
selection		String (cs)	<i>Taglib</i> selection="rb_k1" <i>Classlib</i> setSelection("rb_k1")
verticalAlign		-	<i>Taglib</i> No tag available <i>Classlib</i> setVerticalAlign(CellVAlign radioBCol, CellVAlign textCol)

Events	M	Values	Usage
onClick		String (cs)	<i>Taglib</i> onClick="ProcessClick" <i>Classlib</i> setOnClick("ProcessClick")

Example

using the taglib

Uniform Resource Name (URN)

```
<hbj:radioButtonGroup
  id="Genderselect"
  columnCount="2"
  selection="rb_fem"

  <hbj:radioButton
    id="RBGenderFemale"
    text="female"
    key="rb_fem"
    tooltip="I am female"
    disabled="false"
  />
  <hbj:radioButton
    id="RBGenderMale"
    text="male"
    key="rb_male"
    tooltip="I am male"
    disabled="false"
  />
</hbj:radioButtonGroup>
```

using the classlib

```
Form form = (Form) this.getForm();
RadioButtonGroup rbg = new RadioButtonGroup("Genderselect");
rbg.setColumnCount(2);
RadioButton r1 = rbg.addItem("rb_fem", "female");
RadioButton r2 = rbg.addItem("rb_male", "male");
r1.setTooltip("I am female");
r2.setTooltip("I am male");
r2.setSelected(true);
//      as alternative you can use
//      rbg.setSelection("rb_male");
form.addComponent(rbg);
```

Result

☐ female ☒ male

3.4.2.6.28 Scroll Container**Definition**

The ScrollContainer is an area defined by width and height that displays scrollbars when the contents of the ScrollContainer extends the defined size.

- **height**
Specifies the height of the ScrollContainer control.
- **id**
Identification name of the ScrollContainer control.
- **width**
Specifies the width of the ScrollContainer control.

Uniform Resource Name (URN)

Attributes	M	Values	Usage
height		Unit	<i>Taglib</i> height="300" <i>Classlib</i> setHeight("300")
id	*	String (cs)	<i>Taglib</i> id="scrCont" <i>Classlib</i> setId("scrCont")
width		Unit	<i>Taglib</i> width="500" <i>Classlib</i> setWidth("500")

Example

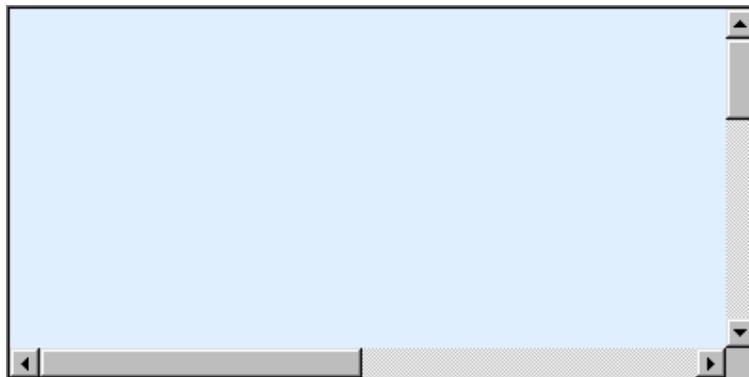
using the taglib

```
<hbj:scrollContainer id="scrCont"  
  width="400"  
  height="200"  
>
```

using the classlib

```
Form form = (Form) this.getForm();  
ScrollContainer scrCont = new ScrollContainer("scrCont");  
scrCont.setWidth("400");  
scrCont.setHeight("200");  
form.addComponent(scrCont);
```

Result



3.4.2.6.29 Table View

Definition

Arrangement of data - text, images, links, other tables etc. - into rows and columns. TableView rows may be grouped into a head, foot and body section. The tableView supplies also navigation buttons which allow browsing thru the table.

TableView example 1						Header
	PLVAR	LOCTP	LOCID	LOCSH	LOCTX	
<input type="checkbox"/>	01	F	50000484	Walldorf	training center Walldorf	
<input type="checkbox"/>	01	F	50000485	Zürich	training center Zürich	
<input type="checkbox"/>	01	F	50000486	Vienna	training center Vienna	
<input type="checkbox"/>	01	F	50000733	Los Angeles	training center Los Angeles	
<input type="checkbox"/>	01	F	50000734	Philadelphia	training center Philadelphia	
						Footer
						1/16

- **cellDisabled**

Enables or disables a control in the specified cell. Currently the inputField control is the only control that can be disabled/enabled.

- **cellHAlignment**

Sets the horizontal alignment of the specified cell. Possible values are:

- LEFT
Left justifies the content of the cell.
- RIGHT
Right justifies the content of the cell.
- CENTER
Centers the content of the cell.
- CHAR
Aligns text around a specific character. Not supported by all web clients.
- JUSTIFY
Sets text in the cell left and right aligned. Not supported by all web clients.

- **cellHeaderVisible**

A boolean value that controls the visibility of the header of the columns. If the value is set to "FALSE" the header of the columns is not rendered.

- **cellInvalid**

This attribute can set the display of a control in the specified cell as "invalid" - to indicate the user that the data in this field is not correct. Currently the inputField control is the only control that can be "invalid".

- **cellRenderer**

Sets a cell renderer for the tableView control.

Uniform Resource Name (URN)

- **cellType**

Sets the display type for the specified cell. The cell type can be following controls:

- BUTTON
- IMAGE
- IMAGELINK
- INPUT
- LINK
- TEXT
- USER
- User defined cell type.

- **cellVAlignment**

Sets the vertical alignment of the specified cell. Possible values are:

- BASELINE
The content of the cell is aligned on the baseline line of the cell (or bottom when no baseline exists).
- BOTTOM
The content of the cell is aligned to the bottom line of the cell.
- MIDDLE
The content of the cell is aligned to the middle of the cell height.
- TOP
The content of the cell is aligned to the top line of the cell.

- **colspanForCell**

Sets the column span (width) of the specified cell.

- **columnAt**

Sets a TableColumn for the specified column.

- **columnCount**

Defines the amount of columns for the tableView.

- **columnHAlignment**

Sets the horizontal alignment of the specified column.

- **columnInvisible**

Sets a column with the specified key invisible.

- **columnName**

Sets the title of the specified column.

- **columnType**

Sets the display type for the specified column. The column type can be following controls:

- BUTTON
- IMAGE

Uniform Resource Name (URN)

- IMAGELINK
 - INPUT
 - LINK
 - TEXT
 - USER
 - User defined column type.
- **columnVAlignment**
Sets the vertical alignment of the specified column.
- **design**
Defines the look of the table
 - ALTERNATING
The rows of the table entries are colored alternating.
 - STANDARD
The background of tableView is uniformly colored.
 - TRANSPARENT
The tableView has no background.
- **emptyTableText**
Sets the text when the table model contains no data - is empty.
- **fillUpEmptyRows**
A boolean value. If set to "TRUE" the tableView has always the height set by the 'visibleRowCount' attribute, regardless of the available table entries. The not available table entries will be filled up with empty lines and according the 'design' attribute.
If set to "FALSE" the tableView height is adjusted to the available table entries.
- **footerRenderer**
Sets a renderer for the table footer.
- **footerVisible**
A boolean value that controls the footer row. If set to "FALSE" the footer row including the navigation buttons is invisible.
- **headerCellRenderer**
Sets a renderer for the table header. The 'headerCellRenderer' renders the header column by column, so you have access to every column header.
- **headerRenderer**
Sets a renderer for the table header.
- **headerText**
Defines the headline of the tableView.
- **headerVisible**
A boolean value that controls the visibility of the header line. If the value is set to "FALSE" the header is not rendered and the table view starts with the cell header row. If the cell header row is invisible also, the table view starts with the data rows.
- **id**

Uniform Resource Name (URN)

Identification name of the tableView.

- **keyColumn**

Sets the specified column as key column.

- **linkColumnKey**

Sets the specified column as link

- **model**

Defines the model which provides the tableView with data. How to setup a [TableViewModel \[Page 33\]](#).

- **navigationMode**

Controls the navigation buttons in the footer row.

- **BYPAGE**

Four navigation buttons are displayed that allow browsing page up, page down, first and last table entry

- **BYLINE**

Two additional buttons are displayed allowing single row up and down.

- **onCellClick**

Defines the event handling method that will be processed when the user clicks on one cell of the tableView. The 'onCellClick' event is set for an entire column. You can specify the column by column index or column key. For details see how to setup and work with the [onCellClick \[Page 33\]](#) event.

- **onClientCellClick**

Defines a Javascript fragment that is executed when the user clicks on one cell of the tableView. The 'onClientCellClick' event is set for an entire column. You can specify the column by column index or column key.

- **onClientRowSelection**

Defines a Javascript fragment that is executed when the user clicks on the radiobutton button in the first column.

- **onHeaderClick**

Defines the event handling method that will be processed when the user clicks on the header of the table. For details see how to setup and work with the [onHeaderClick \[Page 33\]](#) event.

- **onNavigate**

Defines the event handling method that will be processed when the user clicks on the navigation buttons. For details see how to setup and work with the [onNavigate \[Page 33\]](#) event.

- **onRowSelection**

Defines the event handling method that will be processed when the user clicks on the radiobutton button in the first column. The radiobutton is visible when the 'selectionMode' is set to "SINGLESELECT". The method `com.sap.htmlb.event.TableSelectionEvent.getRowIndex()` can be used to retrieve the index of the row that initiated the event. For details see how to setup and work with the [onRowSelection \[Page 33\]](#) event.

- **rowCount**

Uniform Resource Name (URN)

Defines the maximum record that is displayed together with the actual position (for example, 3/16 - meaning: actual position 3, maximum records 16) on the right side of the footer. If 'rowCount' is not specified the number of records in the model define the value.

- **rowRenderer**

Sets a row renderer.

- **rowSelectable**

Defines if the checkbox or radiobutton (dependend on the 'selectionMode') of the specified row, can be selected.

- **rowspanForCell**

Sets the row span for the specified cell.

- **rowVAlignment**

Sets the vertical alignment for a row.

- **selectionMode**

Defines how the table entries can be selected

- MULTISELECT

A checkbox is displayed on every row at the first column of the table. According to the nature of the checkbox multiple columns can be selected at a time.

- NONE

No selection possible (no checkbox and no radiobutton).

- SINGLESELECT

A radiobutton is displayed on every row at the first column of the table. According to the nature of the radiobutton one column can be selected at a time. Together with this attribute 'onRowSelection' attribute can be set so that an event is fired, when the user clicks on the radiobutton.

- **styleForCell**

Sets the style for the specified cell. Style can be BADVALUE, GOODVALUE, CRITICALVALUE to name a few.

- **summary**

Sets the summary.

- **tableHAlignment**

Sets the horizontal alignment for the entire tableView control.

- **userTypeCellRenderer**

Set a user defined cell renderer.

- **visibleFirstRow**

Defines the number of the table entry that is displayed in the first row of the tableView. All subsequent entries are displayed accordingly.

- **visibleRowCount**

Defines the visible rows of the tableView. If 'fillUpEmptyRows' is set to "TRUE", all rows specified with 'visibleRowCount' are displayed. The default for 'visibleRowCount' is the number of table entries supplied by the model.

- **width**

Uniform Resource Name (URN)

Defines the width of tableView.

Attributes	M	Values	Usage
cellDisabled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellDisabled(int rowIndex, int colIndex, boolean state)</code>
cellHAlignment		CENTER CHAR JUSTIFY LEFT RIGHT	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellHAlignment(int rowIndex, int colIndex, CellHAlign.RIGHT)</code>
cellHeaderVisible		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellHeaderVisible(true)</code>
cellInvalid		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellInvalid(int rowIndex, int colIndex, Boolean state)</code>
cellRenderer		Component	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellRenderer (ICellRenderer cr)</code>
cellType		BUTTON IMAGE IMAGELINK INPUT LINK TEXT (d) USER	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellType(int rowIndex, int colIndex, TableColumnType.LINK)</code>
cellVAlignment		BASELINE BOTTOM MIDDLE (d) TOP	<i>Taglib</i> No tag available <i>Classlib</i> <code>setCellVAlignment(int rowIndex, int colIndex, CellVAlign.TOP)</code>
colspanForCell		Numeric (1)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setColspanForCell(int rowIndex, int colIndex,</code>

Uniform Resource Name (URN)

			int colspan)
columnAt		Numeric	<i>Taglib</i> No tag available <i>Classlib</i> setColumnAt (TableColumn col, int colIndex)
columnCount		Numeric (1)	<i>Taglib</i> No tag available <i>Classlib</i> setColumnCount (10)
columnHAlignment		CENTER CHAR JUSTIFY LEFT RIGHT	<i>Taglib</i> No tag available <i>Classlib</i> setColumnHAlignment (int colIndex, CellHAlign.RIGHT)
columnInvisible		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setColumnInvisible ("columnName")
columnName		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setColumnName ("columnName", 2)
columnType		BUTTON IMAGE IMAGELINK INPUT LINK TEXT (d) USER	<i>Taglib</i> No tag available <i>Classlib</i> setColumnType (TableColumnType.LINK, int colIndex)
columnVAlignment		BASELINE BOTTOM MIDDLE (d) TOP	<i>Taglib</i> No tag available <i>Classlib</i> setColumnVlignment (int colIndex, CellVAlign.TOP)
design		STANDARD (d) ALTERNATING TRANSPARENT	<i>Taglib</i> design="STANDARD" <i>Classlib</i> setDesign (TableViewDesign.STANDARD)
emptyTableText		String	<i>Taglib</i> No tag available <i>Classlib</i> setEmptyTableText ("No data")
fillUpEmptyRows		FALSE TRUE (d)	<i>Taglib</i> fillUpEmptyRows="FALSE"

Uniform Resource Name (URN)

			<i>Classlib</i> setFillUpEmptyRows (false)
footerRenderer		Component	<i>Taglib</i> No tag available <i>Classlib</i> setFooterRenderer (IFooterRenderer fr)
footerVisible		FALSE TRUE (d)	<i>Taglib</i> footerVisible="FALSE" <i>Classlib</i> setFooterVisible (false)
headerCellRenderer		Component	<i>Taglib</i> No tag available <i>Classlib</i> setHeaderRenderer (IHeaderRenderer r)
headerRenderer		Component	<i>Taglib</i> No tag available <i>Classlib</i> setHeaderRenderer (IHeaderRenderer r)
headerText		String	<i>Taglib</i> headerText="SAP training" <i>Classlib</i> setHeaderText ("SAP training")
headerVisible		FALSE TRUE (d)	<i>Taglib</i> headerVisible="FALSE" <i>Classlib</i> setHeaderVisible (false)
id	*	String (cs)	<i>Taglib</i> id="tCenter" <i>Classlib</i> setId ("tCenter")
keyColumn		Numeric (defined by model)	<i>Taglib</i> No tag available <i>Classlib</i> setKeyColumn (3)
linkColumnKey			<i>Taglib</i> No tag available <i>Classlib</i> setLinkColumnKey ("linkRef", 3)
model [Page 33]		Component	<i>Taglib</i> model=" myBean.model [Page 33] " <i>Classlib</i> setModel ((TableViewModel [Page 33])

Uniform Resource Name (URN)

			model)
navigationMode		BYPAGE (d) BYLINE	Taglib navigationMode="BYLINE" Classlib setNavigationMode (TableNavigationMode.BYLINE)
rowCount		Numeric (defined by model)	Taglib rowCount="5" Classlib setRowCount (5)
rowRenderer		Component	Taglib No tag available Classlib setRowRenderer (IRowRenderer r)
rowSelectable		FALSE TRUE (d)	Taglib No tag available Classlib setRowSelectable (3, false)
rowspanForCell		Numeric	Taglib No tag available Classlib setRowSpanForCell (int rowIndex, int colIndex, int rSpan)
rowVAlignment		BASELINE BOTTOM MIDDLE (d) TOP	Taglib No tag available Classlib setRowVAlignment (3, CellVAlign.TOP)
selectionMode		MULTISELECT (d) NONE SINGLESELECT	Taglib selectionMode="NONE" Classlib setSelectionMode (TableSelectionMode.NONE)
styleForCell		BADVALUE_DARK BADVALUE_LIGHT BADVALUE_MEDIUM CRITICALVALUE_DARK CRITICALVALUE_LIGHT CRITICALVALUE_MEDIUM GOODVALUE_DARK GOODVALUE_LIGHT GOODVALUE_MEDIUM	Taglib No tag available Classlib setStyleForCell (int rowIndex, int colIndex, TableCellStyle.NEGATIVE)

Uniform Resource Name (URN)

		GROUP _HIGHLIGHTED GROUP _HIGHLIGHTED _LIGHT GROUP_LEVEL1 GROUP_LEVEL2 GROUP_LEVEL3 KEY_MEDIUM MARKED NEGATIVE POSITIVE STANDARD (d) SUBTOTAL SUBTOTAL_LIGHT TOTAL	
summary		String	<i>Taglib</i> No tag available <i>Classlib</i> setSummary("Total")
tableHAlignment		CENTER CHAR JUSTIFY LEFT (d) RIGHT	<i>Taglib</i> No tag available <i>Classlib</i> setTableHAlignment (CellHAlign.CENTER)
userTypeCellRenderer		Component	<i>Taglib</i> No tag available <i>Classlib</i> setUserTypeCellRenderer (ICellRenderer r)
visibleFirstRow		Numeric (1)	<i>Taglib</i> visibleFirstRow="5" <i>Classlib</i> setVisibleFirstRow(5)
visibleRowCount		Numeric (defined by model)	<i>Taglib</i> visibleRowCount="20" <i>Classlib</i> setVisibleRowCount(20)
width		Unit	<i>Taglib</i> width="500" <i>Classlib</i> setWidth("500")

Events	M	Values	Usage
onCellClick [Page 33]		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setOnClick(int colIndex, "Pr_Cell")</code> <code>setOnClick("colKey", "Pr_Cell")</code>
onClientCellClick [Page 33]		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setClientCellClick("alert('Click')")</code>
onClientRowSelection [Page 33]		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setClientRowSelection("alert('Click')")</code>
onHeaderClick [Page 33]		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setOnHeaderClick("onHeaderClickMethod")</code>
onNavigate [Page 33]		String (cs)	<i>Taglib</i> <code>onNavigate("onNavigateMethod")</code> <i>Classlib</i> <code>setOnNavigate("onNavigateMethod")</code>
onRowSelection [Page 33]		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setOnRowSelection("onRowSelMethod")</code>

Example

using the taglib

```

<hbj:tableView
  id="myTableView1"
  model="myTableViewBean.model"
  design="ALTERNATING"
  headerVisible="true"
  footerVisible="true"
  fillUpEmptyRows="true"
  navigationMode="BYLINE"
  selectionMode="MULTISELECT"
  headerText="TableView example 1"
  onNavigate="myOnNavigate"
  visibleFirstRow="1"
  visibleRowCount="5"
  rowCount="16"
  width="500 px"
/>

```

using the classlib (For information about setting up the bean, see [DefaultTableModel](#) [\[Page 33\]](#))

Uniform Resource Name (URN)

```

Form form = (Form) this.getForm();
TableView tv = new TableView("myTableView1");
MyTableViewBean myBean = new MyTableViewBean();
tv.setModel(myBean.getModel());
tv.setDesign(TableViewDesign.ALTERNATING);
tv.setHeaderVisible(true);
tv.setFooterVisible(true);
tv.setFillUpEmptyRows(true);
tv.setNavigationMode(TableNavigationMode.BYLINE);
tv.setSelectionMode(TableSelectionMode.MULTISELECT);
tv.setHeaderText("TableView example 1");
tv.setOnNavigate("myOnNavigate");
tv.setVisibleFirstRow(1);
tv.setVisibleRowCount(5);
tv.setRowCount(16);
tv.setWidth("500 px");
form.addComponent(tv);

```

Result

TableView example 1					
	PLVAR	LOCTP	LOCID	LOCSH	LOCTX
<input type="checkbox"/>	01	F	50000484	Walldorf	training center Walldorf
<input type="checkbox"/>	01	F	50000485	Zürich	training center Zürich
<input type="checkbox"/>	01	F	50000486	Vienna	training center Vienna
<input type="checkbox"/>	01	F	50000733	Los Angeles	training center Los Angeles
<input type="checkbox"/>	01	F	50000734	Philadelphia	training center Philadelphia
					1/16

3.4.2.6.29.1 Events

The TableView component provides several events that can be processed on the client and on the server.

3.4.2.6.29.1.1 onCellClick

The event is fired when the user clicks on one cell of a column. Every column (not cell) can have its own event handling routine.

Example

Uniform Resource Name (URN)

```

public TableViewExample() {
    model = this.createNewTable(model);

    /* getting column name and define event handler for the column */
    TableColumn column = model.getColumn("1stColumn");
    /* Now define event handler name/method - the method
       has to be declared in the DynPage */
    column.setOnCellClick("1stColumn", "onFirstColumnClick");
    /* Do the same for the other 2 columns in the model */
    column = model.getColumn("2ndColumn");
    column.setOnCellClick("2ndColumn", "onSecondColumnClick");
    column = model.getColumn("3rdColumn");
    column.setOnCellClick("3rdColumn", "onThirdColumnClick");
}

```

3.4.2.6.29.1.2 onHeaderClick

The event is fired when the user clicks on one of the columns in the header of the table.

Example

```

/* Get the tableView component */
TableView table = (TableView) this.getComponentByName("myTableView");

/* Set the onHeaderClick event */
table.setOnHeaderClick("onHeaderClick");
..

/* the method onHeaderClick has to be declared in the DynPage.
   Otherwise an exception is thrown ("method not found") when the
   user clicks on the header of the tableView.

   Declaring onHeaderClick: */
public void onHeaderClick(Event event) {
    System.out.println("Header click");
}

```

3.4.2.6.29.1.3 onNavigate

The event is fired when the user clicks on one of the navigation buttons that are displayed in the footer section of the tableView. The method `getFirstVisibleRowAfter` returns the first visible row after the event. Therefore it is not necessary to investigate which navigation button was pressed. This makes the `onNavigate` event handling routine very small.

Example

using the taglib

Uniform Resource Name (URN)

```
// CALLED IF THE NAVIGATION EVENT WAS SEND
public void onNavigation(Event event) {
    if (event instanceof TableNavigationEvent) {
        // Get the navigation event
        TableNavigationEvent tne = (TableNavigationEvent) event;

        // Use getGirstVisibleRowAfter to receive the first visible row
        this.visibleRow = tne.getFirstVisibleRowAfter();
        if (myBean != null) {
            /* just for the first time, when there is no bean.
             Set the visible row in the model - the tableView
             shows the new result after the page is transmitted */
            myBean.setVisibleRow(new Integer(this.visibleRow).toString());
        }
    } else {
        // wrong event...
    }
}
```

3.4.2.6.29.1.4 onRowSelection

The event is fired when the user clicks on the radio button in the first column. The radio button is visible when the 'selectionMode' is set to "SINGLESELECT".

Example

using the taglib

```
import com.sap.htmlb.enum.TableSelectionMode;
..

/* Get the tableView component */
TableView table = (TableView) this.getComponentByName("myTableView");

/* Set the onHeaderClick event */
table.setSelectionMode(TableSelectionMode.SINGLESELECT);
table.setOnRowSelection("onRowSelClick");
..

/* the method onRowSelClick has to be declared in the DynPage.
   Otherwise an exception is thrown ("method not found") when the
   user clicks on the radio button of the tableView.

   Declaring onHeaderClick: */
public void onRowSelClick(Event event) {
    System.out.println("Row selection click");
}
```

onClientRowSelection

The selected row can also be determined on the client by using the `htmlbevent` object.

Uniform Resource Name (URN)

For JS there are a few API calls available to get to the key.

JavaScript Example:

```
htmlbevent.obj.getClickedRowKey();
```

For more information see [JavaScript API \[Page 33\]](#).

3.4.2.6.29.2 Usage & Type

The table view allows to arrange data - text, images, links, other tables etc. - into rows and columns, that is, in a tabular fashion. Table view rows may be grouped into a header, body and footer section. The table view supplies navigation buttons for scrolling the table. In addition, the table view offers mechanisms for single and multiple selection of rows.

Table View					
	Course	Course #	Location	Training Facility	Date
<input type="radio"/>	HTML Basics I	50000484	Walldorf	Training Center Walldorf	12/01/2001
<input type="radio"/>	HTML Basics II	50000485	Zürich	Training Center Zürich	12/01/2001
<input type="radio"/>	Web Design Beginners	50000486	Wien	Training Center Wien	12/01/2001
<input type="radio"/>	Web Design Advanced	50000733	Los Angeles	Training Center Los Angeles	12/01/2001
<input checked="" type="radio"/>	Java Basics	50000734	Philadelphia	Training Center Philadelphia	12/01/2001
<input type="radio"/>	Javascript Basics	50000736	Atlanta	Training Center Atlanta	12/01/2001
					1/3

Figure 1: Example of a table view with different column types and an erroneous input field

Usage

Table views are primarily used as data tables for displaying numeric or non-numeric tabular data. Table views can be read-only or used for data entry. Depending on the usage of the table view, different looks and behaviors can be chosen.

General Usage Tips

Tables are relatively complex screen elements that lead developers to squeezing in lots of information. Keep tables small with respect to the number of columns and rows.

For long tables consider effective filtering methods like the shuffler: These tools effect that only a few rows are displayed and that users need not scroll, or need to scroll only a little bit.

Also, consider alternative presentations, such as charts or graphs - they may reveal relevant information faster.

Look

The table view can be presented in three alternative looks:

- Grid and background - either with alternating row colors or with a uniformly colored background.
- Transparent, that is, without grid and background.

Selection rules for the different table view presentations:

- Grid and uniform background
Use this look preferably for entry tables and for numeric display tables. Use the uniformly colored background also for narrow to medium wide display tables.
- Stripe pattern background
Use the alternative stripe pattern preferably for data entry tables and for wide display tables.
- Transparent background
Use this look preferably for non-numeric display tables, that is, for tables, which display text and/or images.

Table Title and Table Parts

A table view consists of three main parts: a header row, the table view body, and a footer row.

- The table view header contains the table view's title. A table view should have a title if the table is not described elsewhere (for example, by a group or tray title).
- The table view body contains the actual data.
- The table view footer is located in the bottom row; it contains the scroll buttons to the left and an optional text. The footer text may, for example, offer paging information.

The header row as well as the footer row can be hidden. Note that hiding the footer also hides the scroll buttons.

Row and Column Headers

Row and column headers describe data columns and rows. Typically, a table view has only column headers; these describe the different attributes of items that are listed in rows. Row headers can be used, for example, in matrix-like tables, which have both row and column titles.

Cell Content

Table view cells can contain text, images or icons, links, buttons, input fields and dropdown list boxes.

Like stand-alone input fields, input fields in tables can have different attributes, such as required, read-only, or error state.

Row Selection

The table view can be used in two selection modes, if needed:

- single-selection (radio buttons in the first table column)
- multiple-selection (checkboxes in the first table column)

These mode are set by assigning the values SINGLESELECT or MULTISELECT to the attribute selectionMode.

In the default mode of the table view (selectionMode = NONE) users cannot select any rows. Use this mode if there is no need for users to select rows/items. Use single-selection if users shall only select one row, that is, one item, at a time. Use multiple-selection if users can select several options or items in parallel.

Scrolling

The table view offers up to six buttons for scroll functions: First page, Page up, Line up, Line down, Page down, and Last page. The scroll buttons are invisible if the footer is hidden.

Note: Scrollbars are currently not supported in table views.

Technical Info: The Line up/down buttons appear only if the selectionMode attribute has been set to NONE or SINGLESELECT.

Table View Size

Recommendations for the table view size:

Vertical Size

Table views should vertically fit the window or iView they are placed into. As table views can be scrolled through buttons, there should not be no need to use the window's scrollbars.

Make the height of the table view as large as possible with respect to the surrounding container. The larger the table view, the less scrolling is needed (scrolling through buttons is extremely cumbersome..).

Table views should have at least three visible lines - five lines are even better.

Horizontal Size

Table Views should also horizontally fit the window or iView they are placed into. Avoid horizontal scrolling at any price.

Matrix Tables

Matrix tables should have at least 2x2 data cells.

Initial Size and Appearance

Empty tables are not displayed in iViews. In addition, no space is reserved for the table or for a sentence, such as "No entries found".

Exception: Tables where users can immediately enter data should appear in the intended size and with empty lines.

Note: Do not use the transparent design (see below) in this case.

Placement of Buttons for Functions

Location

Place buttons acting on a table view as a whole below the table view and left-aligned with the table. Place the emphasized button to the left if there is one.

Button Groups

Separate button groups by a spacer (non-breaking space).

Functions Referring to Table View Columns, Sorting

Buttons Referring to Columns

Some functions, for examples Sort, refer to certain columns. Place buttons with small icons into the column headers to speed up interaction.

If there are alternative variants of a function (for example, different sort orders or calculations), consider to display only one icon at a time in order to save space, instead of displaying two or more icons in parallel.

Do not use more than three icons in a column header.

Links in Column Headers

If it is evident what a function does, you can also use a link in the column header text.

Functions Referring to Table View Rows

Functions referring to table rows typically refer to the item the data of which are displayed in a row. These functions can either be presented as

- Buttons
- Icons
- Links

Use buttons for most functions. Use icons and links for the following exception cases:

- Icons
Web standards, such as Delete, Info, Help, or Shopping Basket.
- Links
Navigation functions, such as Details (place the link in the name or ID column), additional information and so on.

As a general rule for selecting the correct control, care for the application context and the respective "Web standards".

Table Cells: Links vs. Buttons

Note that links are not self-explaining. Therefore, use links only where their purpose is evident. Add tooltips to links to support users.

See [Links vs. Buttons \[Page 33\]](#) for more information.

Filtering, Shufflers

Offer filters as much as possible. Filters help to reduce the amount of data displayed in a table view. This helps users and improves performance.

Use shufflers for creating filter statements. See the respective section on shufflers in the iView Guidelines.

Placement

Place the shuffler statement above the table view and left-align it.

Reason: A left-aligned shuffler is not hidden from view if the window size is altered.

Button


Use a button labeled Go to start the filtering process. Use events on dropdown list boxes for simple cases only (one dropdown list box with label).

Types

Global Table View Look

The attribute design defines the global look of the table. It can have one of the following values:

- **ALTERNATING**
The rows of the table entries are colored alternating.
- **STANDARD**
The background of Table View is uniformly colored.
- **TRANSPARENT**
The Table View has no background (grid).

Header		
Column 1	Column 2	Column 3
<input type="checkbox"/>	cell 5	cell 6
<input type="checkbox"/>	cell 9	cell 9
		
Footer		


Header		
Column 1	Column 2	Column 3
<input type="checkbox"/>	cell 5	cell 6
<input type="checkbox"/>	cell 8	cell 9
		
Footer		

Figure 2: Table View with grid and patterned background (left) vs. transparent table

Uniform Resource Name (URN)

Cell Style

Individual cells can be given different styles as shown in figure 3; most of them are used for numeric values.

Style	Value
STANDARD	Text
NEGATIVE	-2,000,000.35
POSITIVE	2,000,000.35
TOTAL	2,000,000.35
SUBTOTAL	0
SUBTOTAL_LIGHT	0
BADVALUE_DARK	-2,000,000.35
BADVALUE_MEDIUM	-2,000.35
BADVALUE_LIGHT	-1.35
CRITICALVALUE_DARK	-0.35
CRITICALVALUE_MEDIUM	-0.15
CRITICALVALUE_LIGHT	-0.01
GOODVALUE_DARK	2,000,000.35
POSITIVE	2,000.35
GOODVALUE_LIGHT	20.35
GROUP_HIGHLIGHTED	Use to emphasize group data, never together with critical colors
GROUP_HIGHLIGHTED_LIGHT	Use to emphasize group level 2 data, never together with critical colors
KEY_MEDIUM	Use for Key Values Unique Identifiers
GROUP_LEVEL1	Use to group like a tree level 1
GROUP_LEVEL2	Use to group like a tree level 2
GROUP_LEVEL3	Use to group like a tree level 3
MARKED	Use to show that a ROW is selected

Figure 3: Example table showing different cell styles - click image for larger view

Note: The "GROUP_HIGHLIGHTED" colors should not be used in conjunction with the "CRITICALVALUE" colors.

Cell Content Types

Typically, table cells contain numeric or alphanumeric text. However, there are more cell types available:

- Text: Text cell - the text cannot be edited
- Image: Cell displays an icon or image
- Link: Cell contains a link (reference)
- Button: Cell contains a button
- Input: The cell can be edited

Uniform Resource Name (URN)

- User: The cell contains a dropdown list box

Different TableColumnType settings

Text	Image	Link	Button	Input	User
Hello World		Apple Computer Inc.	Click Me	<input type="text" value="Enter something"/>	Midnight ▾
Good morning		IBM	Click Me	<input type="text" value="Enter something"/>	Midnight ▾
Good morning		IBM	Click Me	<input type="text" value="Enter something"/>	Midnight ▾

Figure 4: Example table showing different cell types

Design-relevant Attributes

The appearance and behavior of tables can be affected by various attributes.

- Header and Footer

The header text can be defined (headerText) and the header as well as the footer be hidden (Boolean attributes headerVisible, footerVisible).

Note: The footer must be visible if the table has to be scrolled, that is, if the table contains more lines than are visible.

- Size, Number of Lines, Initial Appearance

Another set of attributes determines the width of the table view (width), the number of visible rows (visibleRowCount), the first visible row (firstVisibleRow), and the initial appearance of empty rows (Boolean attribute fillUpEmptyRows).

- Selection Mode

Further attributes define how the table entries can be selected: single, multiple, or none (attribute selectionMode, values SINGLESELECT, MULTISELECT, or NONE).

Note: The selection mode also influences the display of scroll buttons, provided the footer is set to visible.

For details see page Control API for Table View.

Related Controls

[Tree View \[Page 33\]](#), [Item List \[Page 33\]](#), [List Box \[Page 33\]](#)

3.4.2.6.29.3 Browser Support & 508

Renders in all supported Browsers.

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the table view control:

Group	Style	IE5 and above	Netscape 4.7

Uniform Resource Name (URN)

Table Styles	Background Color of Standard Table Cell	x	x
	Background Color of Alternating Table Cell	x	x
	Grid Color	x	
	Cell Height	x	
	Cell Padding	x	
	Background Color of Selected Cell	x	x
	Background Color 1 of Grouping Cell	x	x
	Background Color 2 of Grouping Cell	x	x
	Background Color 3 of Grouping Cell	x	x
Table Icons	Background Position	x	
	Height	x	
	Width	x	
	Padding	x	
	URL to "Top" Icon	x	x
	URL to Inactive "Top" Icon	x	x
	URL to "Page Up" Icon	x	x
	URL to Inactive "Page Up" Icon	x	x
	URL to "Up" Icon	x	x
	URL to Inactive "Up" Icon	x	x
	URL to "Down" Icon	x	x
	URL to Inactive "Down" Icon	x	x
	URL to "Page Down" Icon	x	x
	URL to Inactive "Page Down" Icon	x	x
	URL to "Bottom" Icon	x	x
	URL to Inactive "Bottom" Icon	x	x
Container	Font Color of Container Title	x	x
	Background Color of Container Body	x	x

Table 1: Editable styles for the table view control

Accessibility - 508 Support

- Keyboard
Table views are not inserted into the accessibility hierarchy by default.
- Default Description
Is provided by the HTMLB rendering engine for the title, the navigation buttons, and certain elements inside the table.
- Application-specific Description
Set the table title using the `SetHeaderText` method. Set the summary text using the `setSummary` method (should display a tooltip). Note that there is no `setTooltip` method for table views, use `setSummary`, instead.

3.4.2.6.30 Tabstrip

Definition

A container that enables the user to switch between several panels -by clicking on the tab - that appear to share the same space on the screen. The user can view a particular panel by

Uniform Resource Name (URN)

clicking its tab. Use `tabStripItem` to define the panel size and title. Use `tabStripItemBody` to define the layout of the `tabStripItem`. Use `tabStripItemHeader` to change the settings of the title (specified through `tabStripItem`).

- **bodyHeight**

Defines the height of panel. The tabs are added on top of panel. The height of the tabs is defined by used text font.

- **horizontalAlignment**

Defines the horizontal alignment of the `tabStripItems`.

- LEFT

Left justifies the content of the cell.

- RIGHT

Right justifies the content of the cell.

- CENTER

Centers the content of the cell.

- CHAR

Aligns text around a specific character. Not supported by all web clients.

- JUSTIFY

Sets text in the cell left and right aligned. Not supported by all web clients.

- **id**

Identification name of the `tabStrip`.

- **selection**

Defines which tab is the active/displayed panel.

- **tooltip**

Defines the hint of the tab which is displayed as the mouse cursor passes over the panel of the `tabStrip`, or as the mouse button is pressed but not released.

- **verticalAlignment**

Defines the vertical alignment of the `tabStripItems`.

- BASELINE

The content of the cell is aligned on the baseline line of the cell (or bottom when no baseline exists).

- BOTTOM

The content of the cell is aligned to the bottom line of the cell.

- MIDDLE

The content of the cell is aligned to the middle of the cell height.

- TOP

The content of the cell is aligned to the top line of the cell.

- **width**

Defines the overall width of the `tabStrip` control.

Uniform Resource Name (URN)

Attributes	M	Values	Usage
bodyHeight		Unit (100)	<i>Taglib</i> bodyHeight="100" <i>Classlib</i> setBodyHeight("100")
horizontalAlignment		CENTER (d) CHAR JUSTIFY LEFT RIGHT	<i>Taglib</i> horizontalAlignment="LEFT" <i>Classlib</i> setHorizontalAlignment(CellHAlign.LEFT)
id	*	String (cs)	<i>Taglib</i> id="TabbedNotebook" <i>Classlib</i> setId("TabbedNotebook")
selection		Numeric (1)	<i>Taglib</i> selection="3" <i>Classlib</i> setSelection(3)
tooltip		String	<i>Taglib</i> tooltip="select a tab" <i>Classlib</i> setTooltip("select a tab")
verticalAlignment		BASELINE BOTTOM MIDDLE TOP (d)	<i>Taglib</i> verticalAlignment="MIDDLE" <i>Classlib</i> setVerticalAlignment(CellVAlign.MIDDLE)
width		Unit (400)	<i>Taglib</i> width="200" <i>Classlib</i> setWidth("200")

tabStripItem

Specifies the panel size and the tab of a tabStrip. Use tabStripItemBody to define the layout of the tabStripItem. Use tabStripItemHeader to change the settings of the title later on. A tabStripItem must have a unique 'id' and 'index' attribute and can call a specific event handler that is activated when this tab is clicked.

- **header**

The tab can have text (set by the 'title' attribute) or any other control. 'header' specifies the component. Common use would be to display icons in the tabs (instead of text).

- **height**

Defines the height of the tabStripItem.

- **id**

Uniform Resource Name (URN)

Identification name of the `tabStripItem`.

- **index**

Defines the index of the `tabStripItem`. The 'selection' attribute of the `tabStrip` refers to the 'index'. The 'index' is mandatory and can be alphanumeric.

- **onSelect**

Defines the event handling method that will be processed when the user clicks on the tab. The string for the event name is not case sensitive - the reference however has to be spelled exactly the same way as the definition of the 'onSelect' event.

If you define a 'onSelect' event, you have to switch to the new selected tab in your application, using the `tabstrip` method `selection` (see above). If you do not define a 'onSelect' event the tab can be clicked, the selected tab is displayed, but no event is generated.

- **title**

Defines the text that is displayed in the tab itself.

- **tooltip**

Defines the hint of the tab which is displayed as the mouse cursor passes over the panel of the `tabStrip`, or as the mouse button is pressed but not released.

- **width**

Defines the overall width of the `tabStripItem`.

Attributes	M	Values	Usage
header		Component	<i>Taglib</i> No tag available <i>Classlib</i> <pre>setHeader (htmlb.Image ("icon.gif", "Text title")</pre>
height		Unit	<i>Taglib</i> <pre>height="80"</pre> <i>Classlib</i> <pre>setHeight ("80")</pre>
id	*	String (cs)	<i>Taglib</i> <pre>id=" TabbedNotebook "</pre> <i>Classlib</i> <pre>setId ("TabbedNotebook")</pre>
index	*	String (cs)	<i>Taglib</i> <pre>index="I3"</pre> <i>Classlib</i> <pre>setIndex ("I3")</pre>
title		String	<i>Taglib</i> <pre>title="Settings"</pre> <i>Classlib</i> <pre>setTitle="Settings"</pre>
tooltip		String	<i>Taglib</i> <pre>tooltip="Desktop settings"</pre>

Uniform Resource Name (URN)

			<i>Classlib</i> setTooltip("Desktop settings")
width		Unit	<i>Taglib</i> width="200" <i>Classlib</i> setWidth("200")

Events	M	Values	Usage
onSelect		String (cs)	<i>Taglib</i> onSelect="proc_tab3" <i>Classlib</i> setOnSelect("proc_tab3")

tabStripItemBody

Specifies the layout of the tabStripItem.

tabStripItemHeader

The tabStripItem attributes 'title' and 'header' can be altered or set by tabStripItemHeader (see following example definition of "tab 4").

Example

using the taglib

Uniform Resource Name (URN)

```
<hbj:tabStrip
  id="myTabStrip1"
  bodyHeight="100"
  width="200"
  horizontalAlignment="CENTER"
  verticalAlignment="TOP"
  selection="3"
  tooltip="Tooltip for myTabStrip1"
>
<hbj:tabStripItem
  id="myTabStripItem1"
  index="1"
  height="80"
  width="160"
  onSelect="myTabStripItem1OnSelect"
  title="Tab 1"
  tooltip="My Tooltip for Tab 1"
>
  <hbj:tabStripItemBody>
    <hbj:textView
      text="TextView on Tab 1"
    />
  </hbj:tabStripItemBody>
</hbj:tabStripItem>
<hbj:tabStripItem
  id="myTabStripItem2"
  index="2"
  height="80"
  width="160"
  onSelect="myTabStripItem2OnSelect"
  title="Tab 2"
  tooltip="My Tooltip for Tab 2"
>
  <hbj:tabStripItemBody>
    <hbj:textView
      text="TextView on Tab 2"
    />
  </hbj:tabStripItemBody>
</hbj:tabStripItem>
<hbj:tabStripItem
  id="myTabStripItem3"
  index="4"
  height="80"
  width="160"
  onSelect="myTabStripItem3OnSelect"
  tooltip="My Tooltip for Tab 3"
>
  <hbj:tabStripItemBody>
    <%
      myTabStripItem3.setHeader(new
        com.sap.htmlb.Image
        ("/icons/bottom.gif",
        "Image not available")
      );
    %>
    <hbj:textView
      text="TextView on Tab 3"
    />
  </hbj:tabStripItemBody>
</hbj:tabStripItem>
<hbj:tabStripItem
  id="myTabStripItem4"
  index="3"
  height="80"
  width="160"
  onSelect="myTabStripItem4OnSelect"
```

Uniform Resource Name (URN)

using the classlib

```
Form form = (Form) this.getForm();
TabStrip ts = new TabStrip("myTabStrip");
ts.setBodyHeight("100");
ts.setWidth("200");
ts.setHAlign(CellHAlign.CENTER);
ts.setVAlign(CellVAlign.TOP);
ts.setSelection(3);
ts.setTooltip("Tooltip for myTabStrip1");

TextView tvt1 = new TextView();
tv1.setText("TextView on Tab 1");
TextView tvt2 = new TextView();
tv2.setText("TextView on Tab 2");
TextView tvt3 = new TextView();
tv3.setText("TextView on Tab 3");
TextView tvt4 = new TextView();
tv4.setText("TextView on Tab 4");

TabStripItem tsi1 = ts.addTab(1, "Tab 1", tvt1);
tsi1.setHeight("80");
tsi1.setWidth("160");
tsi1.setOnSelect("myTabStripItem1OnSelect");

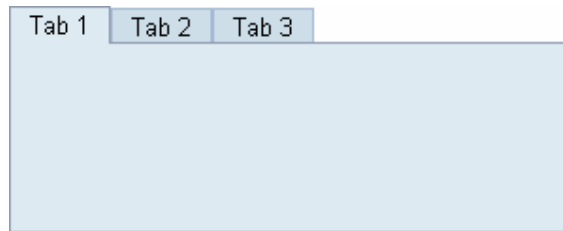
TabStripItem tsi2 = ts.addTab(2, "Tab 2", tvt2);
tsi2.setHeight("80");
tsi2.setWidth("160");
tsi2.setOnSelect("myTabStripItem2OnSelect");
tsi2.setTooltip("My Tooltip for Tab 2");

TabStripItem tsi3 =
    ts.addTab(
        3,
        new Image(
            request.getPublicResourcePath() + "../mimes/saplogo.gif",
            "Logo"),
        tv3);
tsi3.setHeight("80");
tsi3.setWidth("160");
tsi3.setOnSelect("myTabStripItem3OnSelect");
tsi3.setTooltip("My Tooltip for Tab 3");

TabStripItem tsi4 = ts.addTab(4, "Tab 4", tvt4);
tsi4.setHeight("80");
tsi4.setWidth("160");
tsi4.setOnSelect("myTabStripItem4OnSelect");
tsi4.setTooltip("My Tooltip for Tab 4");

form.addComponent(ts);
```

Result



3.4.2.6.30.1 Usage & Type

The tabstrip is a container that allows the user to switch between several views by clicking the tabs. The views appear to share the same space on the screen. The user can access a particular view by clicking its tab.

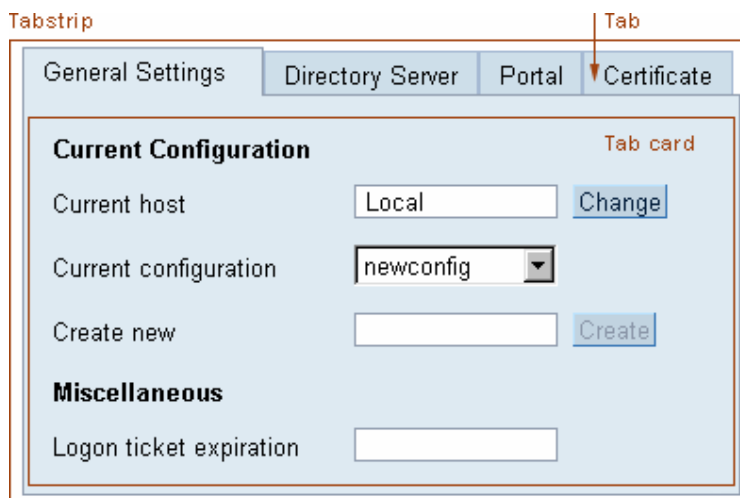


Figure 1: Example of a tabstrip control with an individual tab and the tab card indicated

Usage

Advantages

- Users can see all the alternative views at once. Thus users have a stable context and can navigate easily between the views.
- Tabstrips are also the ideal choice for presenting multiple views of information when the views look very different from one another and a different form of presentation would cause an unstable environment.

Disadvantages

- Tabstrips consume a lot of space compared to other view switching alternatives (for example, radio buttons or dropdown list box shufflers). For views and alternatives to tabstrips that consume less space, see Related Controls.
- Another disadvantage of tabstrips is that the number of views is limited by the space for the tabs.

Do

- Tabstrips may contain dynamic information so that users get a quick overview of important data, events or changes within the views.
- The tab card may contain tables and group boxes.

Don't

- Avoid using long names in the tab labels and using too many tabs, as this will cause the control to be very wide and may cause problems such as scrolling or excessively wide iViews.
- Tabs may not contain icons.
- Tabstrips indicate to the user that views can be accessed in any order; if this is not the case, then avoid using tabstrips.
- Although space is limited in the tab card, it should not be scrolled.
- Tabstrips may not be nested inside one another!

General Usage Tips

Use tabstrips for selecting views only if other alternatives lead to an unstable interface that might confuse users: Tabstrips appear rather massive, and they take a lot of screen real estate. Furthermore tabstrips should only be used for tasks without a prescribed order of steps as they communicate freedom of choice of interaction sequence.

Design-relevant Attributes

The appearance and behavior of tabstrips can be affected by various attributes.

- Height and Width
Attribute `bodyHeight` sets the vertical size of the tabstrip panel, attribute `width` the overall width of the tabstrip.
- Horizontal and Vertical Alignment of Tabs
Use attributes `horizontalAlignment` (values CENTER, CHAR, JUSTIFY, LEFT, RIGHT) and `verticalAlignment` (values BASELINE, BOTTOM, MIDDLE, TOP) to align the tabs.
- Selected Tab
Attribute `selection` selects a tab.

Uniform Resource Name (URN)

- Tooltip Text

Use attribute tooltip to set the tooltip text for a tabstrip as a whole.

In addition for each view (or tabstrip item) several attributes can be set individually:

- Height and Width

Attribute height sets the vertical size of the tabstrip view, attribute width its width.

- Tab Text

Attribute title sets the label for the tab.

- Tooltip Text

Use attribute tooltip to set the tooltip text for a tabstrip view.

For details see page [Control API \[Page 33\]](#) for Tabstrip.

Related Controls

[Radio Button \[Page 33\]](#), [Dropdown List Box \[Page 33\]](#)

3.4.2.6.30.2 Browser Support & 508

Some versions of Netscape Navigator cannot display certain visual nuances of the standard tabstrip control.

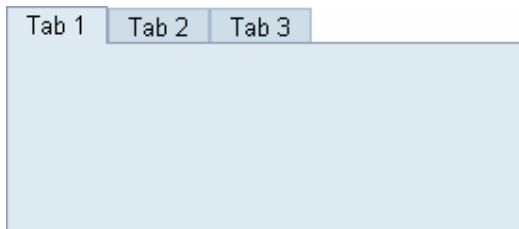


Figure 1: Example of the standard tabstrip

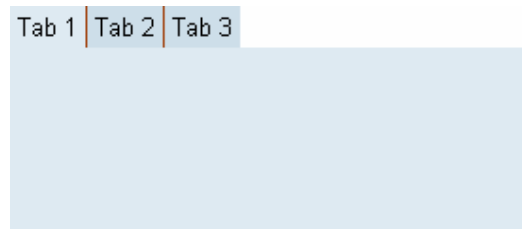


Figure 2: Example of the tabstrip in Netscape Navigator 4

This tabstrip is much less sophisticated visually than the standard tabstrip (figure 1) - the tabs and the tab card have no border, there is a space between the tabs, and the height of the active tab is the same as the height of the inactive tabs.

Uniform Resource Name (URN)

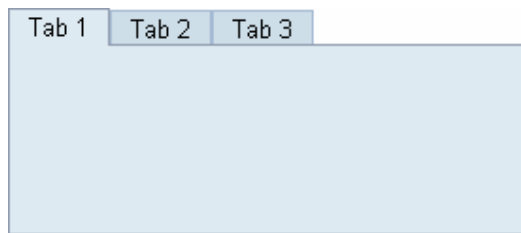


Figure 3: Example of the tabstrip in Netscape Navigator 6.1

This tabstrip is different from the standard tabstrip (figure 1) in that there is a space between the tabs. Additionally, the active tab is the same height as the inactive tabs.

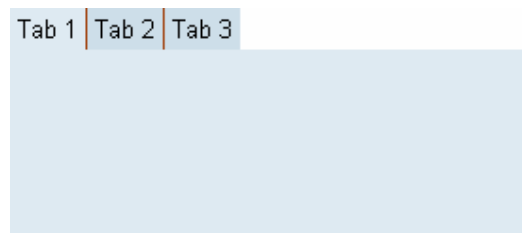


Figure 4: Example of the tabstrip in Netscape Navigator 6.2

This tabstrip is different from the standard tabstrip (figure 1) in that the tabs are all the same height.

Tabstrip Items

Tabstrip items cannot be stored on the Web client. The application has to manage tabstrip items. Therefore, changing the tabs always generates the event `tabSelectionChange`. It is recommended to at least declare the method so that no exception will be thrown if the application is opened in a Netscape Navigator 4 Web client.

Editability in Style Editor

In the Style Editor, it is possible to change all the background and border colors, as well as the padding and all the text attributes. Here is a list of the styles you can influence:

Group	Style	IE5 and above	Netscape 4.7
Tabstrip Styles	Background Color of Selected Tab	x	x
	Background Color of Inactive Tab	x	
	Left Border of Inactive Tabs	x	
	Right Border of Inactive Tabs	x	
	Top Border of Inactive Tabs	x	
	Tab Padding	x	
	Tabstrip Border Color	x	x
	Tab Height	x	x
Container	Container Border	x	
	Top Border of Container	x	
	Right Border of Container	x	
	Left Border of Container	x	

Table 1: Editable styles for the tabstrip control

Accessibility - 508 Support

- **Keyboard**
Each tab of a tabstrip is inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine for each tab.
- **Application-specific Description**
Set an additional description using the `setTooltip` method for each tab if needed.

3.4.2.6.31 Text Edit

Definition

A multiline region for displaying and editing text. Text in the control is restricted to a single font, size and style unless set with HTML commands. For sophisticated text editing see control [htmlEdit \[Page 33\]](#).

The `textEdit` control has a frame. The size of the frame is defined by the 'rows' and 'cols' attribute. A vertical scrollbar is displayed permanently. The scroll bar is enabled when the number of lines exceeds the 'rows' attribute.

If the 'wrapping' attribute is not "OFF" the text is wrapped according to the 'cols' attribute - no horizontal scrollbar. If 'wrapping' is set to "OFF" a horizontal scrollbar is activated if the text line exceeds the width set by the 'cols' attribute.

- **cols**
Defines width of the `textEdit` control in characters. If the text line exceeds the width defined with the 'cols' attribute and the 'wrapping' attribute if "OFF" a horizontal scrollbar is activated. The scrollbar is appended to the `textEdit` frame, so that the 'rows' attribute stays unchanged.

If 'wrapping' is set to "HARD" or "SOFT" the text line is wrapped and no horizontal scrollbar is activated.

Be aware that the definition of 'cols' by characters is only an approach and varies by the used character font. A character font with unequal spacing shows different results. A `i` character fits more often into the `textEdit` field than a `m` character.
- **enabled**
A boolean value that enables or disables the `textEdit` control. A disabled label has a different text color to show the user that it is disabled.
- **height**
Set the height of the text edit control.
- **id**
Identification name of the `textEdit`.
- **labeled**
Enables or disables the notification when the control has a label assigned to it.
- **rows**

Uniform Resource Name (URN)

Defines the height of the textEdit control in lines. If the text lines exceed the 'rows' attribute the vertical scrollbar becomes active.

- **text**

Defines the string of text displayed. This text can be edited and/or new text can be added.

- **tooltip**

Defines the hint of the textEdit which is displayed as the mouse cursor passes over the textEdit, or as the mouse button is pressed but not released.

- **width**

Set the width of the text edit control.

- **wrapping**

Controls the text flow. "HARD" and "SOFT" are passed on to the HTML-Output and control how the carriage return is handled. Web clients handle text wrapping differently. Therefor the following description cannot be guaranteed on all web clients.

- HARD

Wraps the text at the width set by the 'cols' attribute. A carriage control is transmitted at every line break.

No horizontal scrollbar is displayed.

- SOFT

Wraps the text at the width set by the 'cols' attribute. No carriage control is transmitted.

No horizontal scrollbar is displayed.

- OFF

The text line is not wrapped. If the text line length exceeds the width set by the 'cols' attribute a horizontal scrollbar is displayed.

Attributes	M	Values	Usage
cols		Numeric (35)	<i>Taglib</i> cols="20" <i>Classlib</i> setCols("20")
enabled		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEnabled(true)
height		Unit	<i>Taglib</i> No tag available <i>Classlib</i> setHeight("300px")
id	*	String (cs)	<i>Taglib</i> id="Edit_text" <i>Classlib</i> setId("Edit_text")

Uniform Resource Name (URN)

labeled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setLabelled(true)
rows		Numeric (5)	<i>Taglib</i> rows="10" <i>Classlib</i> setRows(10)
text		String	<i>Taglib</i> text="editable text" <i>Classlib</i> setText("editable text")
tooltip		String	<i>Taglib</i> tooltip="PDK document" <i>Classlib</i> setTooltip("PDK document")
wrapping		HARD (d) SOFT OFF	<i>Taglib</i> wrapping="OFF" <i>Classlib</i> setWrapping(TextWrapping.SOFT)

Example

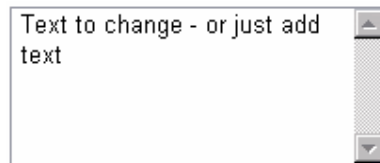
using the taglib

```
<hbj:textEdit
  id="Edit_Text
  text="Text to change - or just add text"
  wrapping="SOFT"
  tooltip="Edit and/or add text"
  rows="10"
  cols="30"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
TextEdit te = new TextEdit("Edit_Text");
te.setText("Text to change - or just add text");
te.setWrapping(TextWrapping.SOFT);
te.setTooltip("Edit and/or add text");
te.setRows(10);
te.setCols(30);
form.addComponent(te);
```

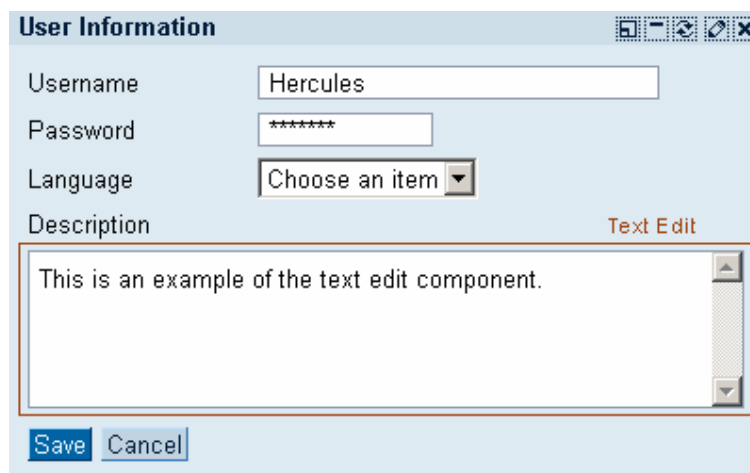
Uniform Resource Name (URN)

Result

Text to change - or just add
text

3.4.2.6.31.1 Usage & Type

The text edit control provides an area for multiple-row text editing.



User Information

Username: Hercules

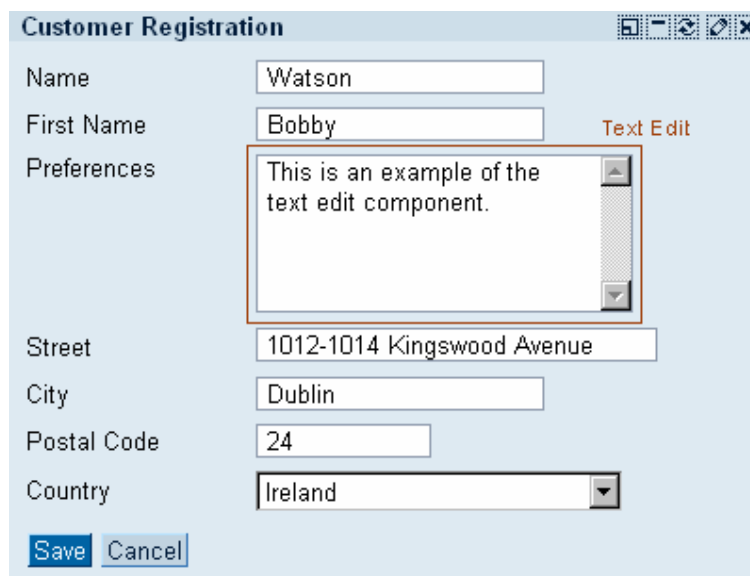
Password: *****

Language: Choose an item

Description: This is an example of the text edit component. Text Edit

Save Cancel

Figure 1: Example of text edit control in an iView



Customer Registration

Name: Watson

First Name: Bobby Text Edit

Preferences: This is an example of the text edit component.

Street: 1012-1014 Kingswood Avenue

City: Dublin

Postal Code: 24

Country: Ireland

Save Cancel

Figure 2: Example of text edit control in an iView

Usage

Use the text edit control to allow users to edit multiple line of text.

The text is restricted to a single font, size and style unless set with HTML commands. The text edit control has a frame. The size of the frame is defined by the rows and cols attributes. A vertical scrollbar is displayed permanently. The scrollbar is enabled when the number of text lines exceeds the number of visible lines.

Alignment

There are two possible ways to align the text edit control:

- Below a group of fields with a descriptive label above the text edit control (figure 1)
- In line with other fields within a field group with a label to the left (figure 2)

Place the text edit below the field group if it is used as the main information, whereas the fields above it provide only the context for the information in the text edit control.

Example: A problem description when sending a problem message to a service center.

Place the text edit field within the field group if the information is just one piece of information among other information, and the text edit control is used as a freeform multiple-line input field.

Example: A customer enters his or her preferences when registering for an online shop.

Design-relevant Attributes

The text edit control can be influenced through a number of attributes:

- The text and a tooltip text can be set (attributes text and tooltip)
- The number of rows and columns can be set (attributes rows and cols)
- The wrapping behavior can be determined (attribute wrapping, values HARD, SOFT and OFF)

For details see [Control API \[Page 33\]](#) for the text edit control.

Related Controls

[Text View \[Page 33\]](#)

3.4.2.6.31.2 Browser Support & 508

In Netscape 4.7 the border will be displayed in 3D; the background color is always white; disabled looks like enabled.

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the text edit control:

Group	Style	IE5 and above	Netscape 4.7
Text Edit Styles	Padding	x	
Container	Container Border	x	

Table 1: Editable styles for the text edit control

Accessibility - 508 Support

Text edit controls have to be used in combination with the label element which points to the assigned text edit control. This ensures, that screenreaders are aware of the relationship between the both elements and can read the correct label to the according text edit control.

- **Keyboard**
Text edit controls are inserted into the accessibility hierarchy by default.
- **Default Description**
Is provided by the HTMLB rendering engine.
- **Application-specific Description**
Set an additional description using the `setTooltip` method if needed.
- **Label**
Has to be connected to a label control (use method `setLabelFor` for identifying the corresponding text edit control).

3.4.2.6.32 Text View

Definition

A multiline region for displaying text. Text in the control is restricted to a single font, size and style unless set with HTML commands.

- **design**
Defines the appearance of the text. Design can be set to "HEADER1", "EMPHASIZED", "LABEL" and so on. The CSS controls how the different options get rendered. The following description is based on the standard CSS delivered.
 - **Emphasized**
Bold text, text size STANDARD
 - **Header 1**
Bold text, text size +4 in comparison to STANDARD
 - **Header 2**

Uniform Resource Name (URN)

Bold text, text size +2 in comparison to STANDARD

- **Header 3**

Bold text, text size STANDARD

- Standard

Text size and attributes STANDARD

- Legend

Text size -2 in comparison to STANDARD

- *Reference*

Italic text, text size STANDARD

- Standard

No text attributes and standard text size

- **encode**

A boolean value that defines how the text is interpreted. HTML text formatting commands (for example, <h1>, <i> etc.) can be used to change the display of the text. If there are no formatting commands in the text string, the encode attribute has no effect.

Example:

```
text="<h1><i>Important</i></h1>"
```

encode = "false" Browser output:

Important

the text string is rendered by interpreting the formatting commands.

encode = "true" Browser output: <h1><i>Important</i></h1>

the formatting commands are displayed and not interpreted.

- **hoverMenuId**

Defines which hover menu is displayed for this tree node. You can define different trigger methods to display the hover menu. For more details, see hover menu.

- **id**

Identification name of the text.

- **labeled**

Enables or disables the notification when the control has a label assigned to it.

- **layout**

Defines the alignment of the text.

- BLOCK

Renders the textView with a <div> HTML tag.

- NATIVE

Renders the textView with a HTML tag.

Uniform Resource Name (URN)

- PARAGRAPH

Renders the textView with a <p> HTML tag

- **required**

Deprecated - the control 'label' should be used instead to label required input fields.

A boolean value. If set to "true" an asterisks (*) in red color is placed at the end of the text string. This is a common method to indicate that input is required. See also `inputField` and `label`.

- **text**

Defines the string of text displayed. See 'encode' for a formatting example with embedded HTML commands.

- **tooltip**

Defines the hint of the textView which is displayed as the mouse cursor passes over the textView, or as the mouse button is pressed but not released.

- **width**

Defines the width of the textView. The width shows only effect when the 'wrapping' attribute is set to "true". Otherwise the width and layout follows the HTML commands in the text string.

- **wrapping**

A boolean value. If set to "true" the text is word wrapped at the set 'width' or - if no 'width' is set - at the form width.

Attributes	M	Values	Usage
design		EMPHASIZED HEADER1 HEADER2 HEADER3 LABEL LABELSMALL LEGEND REFERENCE STANDARD EMPHASIZED HEADER1 HEADER2 HEADER3 LABEL LABELSMALL LEGEND REFERENCE STANDARD (d)	<i>Taglib</i> design="HEADER1" <i>Classlib</i> setDesign (TextViewDesign.HEADER1)
encode		FALSE TRUE (d)	<i>Taglib</i> enabled="FALSE" <i>Classlib</i> setEnabled (false)
hoverMenuId		String (cs)	<i>Taglib</i> hoverMenuId="textHover1" <i>Classlib</i>

Uniform Resource Name (URN)

			setHoverMenuId(textHover1)
id	*	String (cs)	<i>Taglib</i> id="Intro_text" <i>Classlib</i> setId("Intro_text")
labeled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setLayout (TextViewLayout.BLOCK)
required Deprecated		FALSE (d) TRUE	<i>Taglib</i> required="TRUE" <i>Classlib</i> setRequired(true)
text		String	<i>Taglib</i> text="PDK introduction" <i>Classlib</i> setText("PDK introduction")
tooltip		String	<i>Taglib</i> tooltip="PDK document" <i>Classlib</i> setTooltip("PDK document")
width		Unit (100%)	<i>Taglib</i> width="300" <i>Classlib</i> setWidth("300")
wrapping		FALSE (d) TRUE	<i>Taglib</i> wrapping="TRUE" <i>Classlib</i> setWrapping(true)

Example

using the taglib

```
<hbj:textView
  id="Text_ZIP"
  text="ZIP Code"
  design="EMPHASIZED"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
TextView tv2 = new TextView("tv2");
tv2.setText("ZIP Code");
tv2.setDesign(TextViewDesign.EMPHASIZED);
form.addComponent(tv2);
```

Result

ZIP Code

3.4.2.6.32.1 Usage & Type

The text view control offers a multiline region for displaying text; several text attributes can be defined.

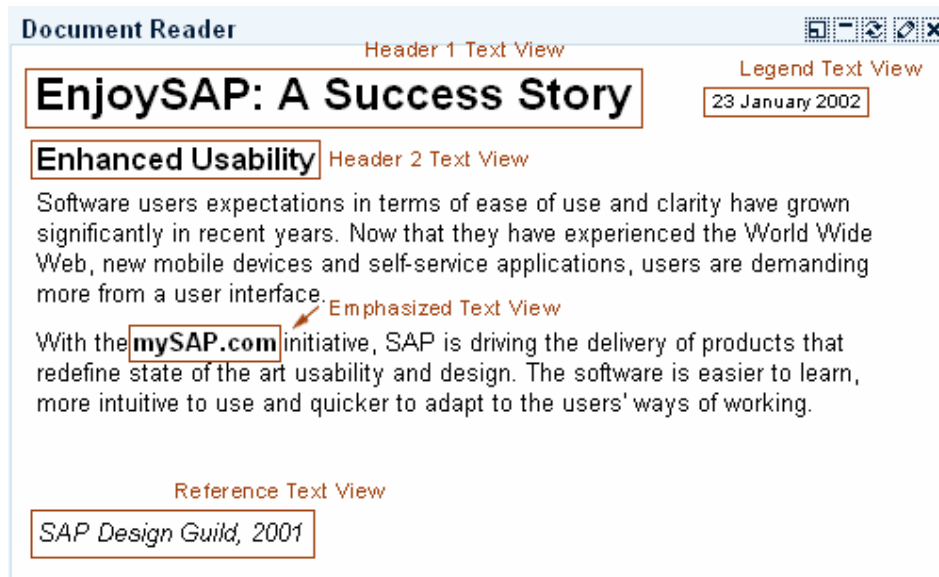


Figure 1: Example of a text view control in an iView.

Usage

The text view control is used to display plain text. The text in the control is restricted to a single font, size and style unless set with HTML commands. The text size can be set using different styles (see "Types of Text Views")

Note: The text View control must not be used to create a label for input fields; use the [label \[Page 33\]](#) control instead.

Also note that if you occupy a certain area on the screen for a text view control you should reserve enough space for the translation to other languages. Text in other languages may use up to 30% more space than needed in English.

Types

The text view control is available in several text styles, which are set by the attribute design (values STANDARD, EMPHASIZED, REFERENCE, LEGEND, HEADER1, HEADER2, HEADER3). The following description is based on the standard CSS delivered:

Text Style	Use
------------	-----

Standard	Used to display body text.
Emphasized	Used to display emphasized text for example, phrases, single terms; not to be used for complete text areas; text size "Standard".
<i>Reference</i>	Style for Text Used as a Reference; text size "Standard".
Legend	Used to display a legend or small-size help text; text size -2 in comparison to "Standard".
Header 1	Used to display a headline or page title; text size +4 in comparison to "Standard"
Header 2	Used to display a page subtitle; text size +2.
Header 3	Used to display a subtitle; text size "Standard".

Table 1: Text styles and their use

Design-relevant Attributes

While the different text types are set using the attribute design (values STANDARD, EMPHASIZED, REFERENCE, LEGEND, HEADER1, HEADER2, HEADER3), the appearance of the different text types can be determined by a style sheet (CSS).

The text itself is set by the attribute text, an accompanying tooltip text by the attribute tooltip.

In addition, alignment (attribute layout, values BLOCK, NATIVE, PARAGRAPH), wrapping behavior (Boolean attribute wrapping), and width (attribute width) can be defined for the text view control.

For details see the [Control API \[Page 33\]](#) for Text View.

Related Controls

[Text Edit \[Page 33\]](#), [Label \[Page 33\]](#)

3.4.2.6.32.2 Browser Support & 508

Renders in every supported browser.

Editability in Style Editor

For the text view control, only common styles can be changed. As these styles are important for the text view control, we list them here, too.

Style Group	Style	IE5 and above	Netscape 4.7
Text Styles	Standard Font Family	x	x
Standard Text	Standard Font Size	x	x
	Standard Font Color	x	x
	Standard Font Style	x	x
	Standard Font Weight	x	x

Non-Standard Text	Font Size for Small Text	x	x
	Font Size for Large Text	x	x
	Font Size for Extra Large Text	x	x
	Font Style for Text Used as Reference	x	x
	Font Color for Headlines	x	x
	Font Weight for Headlines	x	x
	Font Weight for Emphasized Text	x	x

Table 1: Common styles for the text view control

Accessibility - 508 Support

- Keyboard
Text view controls are not inserted into the accessibility hierarchy by default.
- Default Description
Not needed
- Application-specific Description
Not needed

3.4.2.6.33 Tool Bar

Definition

A toolbar provides quick and convenient access to a set of frequently used commands or options. A toolbar can contain buttons, input fields and dropdown list boxes. The toolbar offers a separator. The separator is a vertical line with padding on both sides and is used to space command groups.

The width of the toolbar is defined by the added controls (button, input field and so on). When the toolbar control is not used in a gridlayout or formlayout, it is always rendered in a new line.

- **addToolBarButton**
Adds a button to the toolbar. The toolbar button is similar to the "button" control but has less attributes. See [ToolBarButton \[Page 33\]](#) description for more details.
- **addToolBarDropDownListBox**
Adds a dropdown list box to the toolbar. See [ToolBarDropDownListBox \[Page 33\]](#) description for more details.
- **addToolBarInputField**
Adds an input field to the toolbar. See [ToolBarInputField \[Page 33\]](#) description for more details.
- **addToolBarSeperator**
Adds a vertical line with padding on both sides. The toolbar separator is used to space command groups. See [ToolBarSeparator \[Page 33\]](#) description for more details.
- **design**

Uniform Resource Name (URN)

Sets the design of the toolbar. Possible values are:

- EMPHASIZED
Displays a toolbar with darker background.
- STANDARD
Displays a toolbar with lighter background.

- **id**

Identification name of the toolbar.

Attributes	M	Values	Usage
addToolBarButton		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> addToolBarButton("id", "text")
addToolBarDropDownListBox	*	String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> addToolBarDropDownListBox("id")
addToolBarInputField		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> addToolBarInputField("id")
addToolBarSeparator		-	<i>Taglib</i> No tag available <i>Classlib</i> addToolBarSeparator()
design		EMPHASIZED STANDARD	<i>Taglib</i> design="STANDARD" <i>Classlib</i> setDesign(ToolBarDesign.STANDARD)
id	*	String (cs)	<i>Taglib</i> id="tbl" <i>Classlib</i> Object id.

Example

using the taglib

Uniform Resource Name (URN)

```
<hbj:toolbar
  id="toolbar"
  design="EMPHASIZED">
  <hbj:toolbarButton
    id="OpenButton"
    text="Open"
    onClick="ProcessOpen"
  />
  <hbj:toolbarButton
    id="CloseButton"
    text="Close"
    onClick="ProcessClose"
  />
  <hbj:toolbarSeparator
    id="mySeparator"
  />
  <hbj:toolbarInputField
    id="Scale"
    value="100%"
    width="50px"
  />
  <hbj:toolbarSeparator
    id="mySeparator"
  />
  <hbj:toolbarDropDownListBox
    id="myDDL2">
    <hbj:listBoxItem
      key="k1"
      value="Arial"
    />
    <hbj:listBoxItem
      key="k2"
      value="Times Roman"
    />
    <hbj:listBoxItem
      key="k3"
      value="Verdana"
      selected="true"
    />
  </hbj:toolbarDropDownListBox>
</hbj:toolbar>
```

using the classlib

Uniform Resource Name (URN)

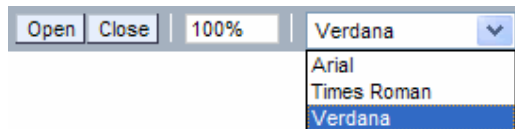
```

Form form = (Form) this.getForm();
Toolbar tb = new Toolbar("toolbar");
tb.setDesign(ToolbarDesign.EMPHASIZED);
ToolbarButton openButton = tb.addToolbarButton("OpenButton", "Open");
openButton.setOnClick("ProcessOpen");
ToolbarButton closeButton = tb.addToolbarButton("CloseButton", "Close");
closeButton.setOnClick("ProcessClose");
tb.addToolbarSeparator();
// Create input field
ToolbarInputField input = new InputField("Scale");
input.setType(DataType.STRING);
input.setWidth("50px");
input.setValue("100%");
tb.addToolbarInputField("Scale");

tb.addToolbarSeparator();
// Create dropdown list box
ToolbarDropDownListBox ddl = new ToolbarDropDownListBox("Fonts");
ddl.addItem("k1", "Arial");
ddl.addItem("k2", "Times Roman");
ddl.addItem("k3", "Verdana");
ddl.setSelection("k3");
tb.addToolbarDropDownListBox("Fonts");

form.addComponent(tb);

```

Result**3.4.2.6.33.1 Tool Bar Button**

Provides any type of functionality in your application at the touch of the button. Hints can be displayed as the mouse cursor passes over the button, or as the mouse button is pressed but not released. The toolbar button has to be used together with the "[toolbar \[Page 33\]](#)" control.

- **enabled**

A boolean value that defines if the toolbar button is clickable. If the toolbar button is disabled (enabled="FALSE") it sends no event when you press a mouse button on the toolbar button. A disabled toolbar button has a different text color to show the user that it is disabled.

- **id**

Identification name of the toolbar button.

- **onClick**

Defines the event handling method that will be processed when the user clicks on the enabled button. If you do not define an 'onClick' event the button can be clicked but no event is generated.

- **onClientClick**

Uniform Resource Name (URN)

Defines the JavaScript fragment that is executed when the user clicks on the button. If both events ('onClick' and 'onClientClick') are specified, the 'onClientClick' event handling method is activated first. By default the 'onClick' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onClick' event handling method with the command `htmlbevent.cancelSubmit=true;`

The 'onClientClick' event is useful to pre-process the form and only send the form to client if the pre-processing was successful (for example, date validation, valid number format and so on) to save client/server interaction.



A button click usually activates the client/server interaction. If an input field has to be filled out for further processing, the JavaScript fragment can check the necessary input on the client side and display a message if the necessary input is missing, without server interaction.



To use JavaScript the JSP has to use the page tag (set [page \[Page 33\]](#) tag).

- **text**

Defines the string of text placed centered on the button. If no text should be displayed in the button an empty string (null) can be used. The width of the button is automatically adjusted to the length of the text.

- **tooltip**

Defines the hint of the button which is displayed as the mouse cursor passes over the button, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
enabled		FALSE (d) TRUE	<i>Taglib</i> <code>enabled="FALSE"</code> <i>Classlib</i> <code>setEnabled(false)</code>
id	*	String (cs)	<i>Taglib</i> <code>id="OrderConfirm"</code> <i>Classlib</i> <code>setId("OrderConfirm")</code>
text		String	<i>Taglib</i> <code>text="Confirm"</code> <i>Classlib</i> <code>setText("Confirm")</code>
tooltip		String	<i>Taglib</i> <code>tooltip="Confirm order"</code> <i>Classlib</i> <code>setTooltip("Confirm order")</code>

Events	M	Values	Usage
--------	---	--------	-------

Uniform Resource Name (URN)

onClick		String (cs)	<i>Taglib</i> onClick="ProcessConfirm" <i>Classlib</i> setOnClick("ProcessConfirm")
onClientClick		String (cs)	<i>Taglib</i> onClientClick="alert('Confirm')" <i>Classlib</i> setOnClientClick("alert('Confirm')")

Example

using the taglib

```
<hbj:toolbarButton id="OpenButton"
  text="Open"
  onClick="ProcessOpen"
/>
```

using the classlib

```
ToolbarButton openButton = tb.addToolbarButton("OpenButton", "Open");
openButton.setOnClick("ProcessOpen");
```

Result

The toolbar button has to be used in the toolbar control. See the "[toolbar \[Page 33\]](#)" control description for the result.

3.4.2.6.33.2 Tool Bar Dropdown List Box

A control with a dropdown arrow that the user clicks to display a list of options. An item in the toolbarDropDownListBox is called listBoxItem. The toolbarDropDownListBox has to be used together with the "[toolbar \[Page 33\]](#)" control.

- **enabled**

A boolean value that defines if the toolbarDropDownListBox is clickable. If the toolbarDropDownListBox is disabled (enabled="FALSE") it is not selectable. A disabled toolbarDropDownListBox has a different color for the displayed listBoxItem.

- **id**

Identification name of the toolbarDropDownListBox.

- **model**

Defines the model which provides the toolbarDropDownListBox with data. How to work with the [IListModel \[Page 33\]](#).

- **nameOfKeyColumn**

Specifies the name of the column that contains the key. This is used when you use an underlying table in the model.

Uniform Resource Name (URN)

- **nameOfValueColumn**

Specifies the name of the column that contains the visible text. This is used when you use an underlying table in the model.

- **onClientSelect**

Defines the JavaScript fragment that is executed when the user clicks on the toolbarDropDownListBox. If both events ('onSelect' and 'onClientSelect') are specified, the 'onClientSelect' event handling method is activated first. By default the 'onSelect' event handling method is activated afterwards. In the JavaScript fragment you can cancel the activation of the 'onSelect' event handling method with the command

```
htmlbevent.cancelSubmit=true;
```

The 'onClientSelect' event is useful to preprocess the form and only send the form to client if the preprocessing was successful (for example, date validation, valid number format etc.) to save client/server interaction.



A toolbarDropDownListBox click usually activates the client/server interaction. If an input field has to be filled out for further processing, the JavaScript fragment can check the necessary input on the client side and display a message if the necessary input is missing, without server interaction.



To use JavaScript the JSP has to use the page tag (see [page \[Page 33\]](#) tag).

- **onSelect**

Defines the event handling method that will be processed when the user clicks on the enabled toolbarDropDownListBox. If you do not define a onClick event the toolbarDropDownListBox can be clicked but no event is generated.

- **selection**

Specifies the key of the listBoxItem which is displayed in the toolbarDropDownListBox.

- **width**

Defines the width of the toolbarDropDownListBox in pixel or percent.

Attributes	M	Values	Usage
enabled		TRUE (d) FALSE	<i>Taglib</i> disabled="TRUE" <i>Classlib</i> setEnabled (false)
id	*	String (cs)	<i>Taglib</i> id = "listbox_te" <i>Classlib</i> setId ("listbox_te")
model [Page 33]		String	<i>Taglib</i> model = " mybean.model [Page 33] "

Uniform Resource Name (URN)

			Classlib setModel(IListModel [Page 33] model)
nameOfKeyColumn		String	Taglib nameOfKeyColumn = "k1" Classlib setNameOfKeyColumn ("k1")
nameOfValueColumn		String	Taglib nameOfValueColumn = "v1" Classlib setNameOfValueColumn ("v1")
selection		String	Taglib selection = "HD" Classlib setSelection("HD")
width		Unit	Taglib width = "200" Classlib setWidth ("200")

Events	M	Values	Usage
onClientSelect		String (cs)	Taglib onClientSelect="alert('Click')" Classlib setOnClientSelect("alert('Click')")
onSelect [Page 33]		String (cs)	Taglib onSelect="proc_listbox" Classlib setOnSelect ("proc_listbox")

listBoxItem

Defines the items in a toolbarDropDownListBox, toolbarDropDownListBox or listBox instead of the model. See [listBoxItem \[Page 33\]](#) for more details.

Example

using the taglib

```
<hbj:toolbarDropDownListBox id="myDDL2" >
  <hbj:listBoxItem key="k1" value="Arial" />
  <hbj:listBoxItem key="k2" value="Times Roman" />
  <hbj:listBoxItem key="k3" value="Verdana" selected="true" />
</hbj:toolbarDropDownListBox>
```

using the classlib

```

ToolbarDropDownListBox ddl = new ToolbarDropDownListBox("Fonts");
ddl.addItem("k1", "Arial");
ddl.addItem("k2", "Times Roman");
ddl.addItem("k3", "Verdana");
ddl.setSelection("k3");

```

Result

The toolbar dropdown list box has to be used in the toolbar control. See the "[toolbar \[Page 33\]](#)" control description for the result.

3.4.2.6.33.3 Tool Bar Input Field

An framed area that allows user input in a toolbar. The toolbarInputField can be displayed with default input. A user can type new text or edit the existing text.

- **enabled**

A boolean value that defines if the inputField allows input. A disabled inputField (enabled = "FALSE") has a different background color.

- **design**

Defines the size of the input field. The value for this attribute can be "STANDARD" or "SMALL".

- **id**

Identification name of the inputField.

- **maxlength**

Defines the maximum amount of characters allowed for the inputField. If the type attribute is set for example, to date or time the 'maxlength' has to take care of the characters delivered by this format and local settings.

- **type**

If 'type' is set to date a help button to call the dateNavigator can be generated (see 'showHelp'). Other than that this attribute has no further effect on the client side. It can be used later on when the form gets processed.

Note: The type INTEGER is not `null` save and will therefore cause an exception when the field is empty. We recommend to use type STRING instead.

- **value**

Default string that is displayed in the inputField frame. The 'maxlength' attribute has no effect on the 'value' attribute. The 'value' string is not truncated to 'maxlength'.

- **width**

Defines the width of the inputField in pixel or percent. This attribute allows better adjustment of the inputField in a form.

The inputField width can also be set by the attribute 'width'. If 'size' and 'width' are set the 'width' attribute has priority and overwrites the 'size' setting.

Attributes	M	Values	Usage
enabled		FALSE TRUE (d)	<i>Taglib</i> enabled="TRUE"

Uniform Resource Name (URN)

			<i>Classlib</i> setEnabled(false)
id	*	String (cs)	<i>Taglib</i> id="GetInput" <i>Classlib</i> setId("GetInput")
invalid		FALSE (d) TRUE	<i>Taglib</i> invalid="TRUE" <i>Classlib</i> setInvalid(true)
maxlength		Numeric (25)	<i>Taglib</i> maxlength="25" <i>Classlib</i> setMaxlength(25)
size		Numeric (30)	<i>Taglib</i> size="35" <i>Classlib</i> setSize("35")
type		BCD BOOLEAN DATE INTEGER STRING TIME	<i>Taglib</i> type="INTEGER" <i>Classlib</i> setType(DataType.INTEGER)
value		String	<i>Taglib</i> value="Your name here" <i>Classlib</i> setValue("Your name here")
width		Unit	<i>Taglib</i> width="200" <i>Classlib</i> setWidth("200")

Example

using the taglib

```
<hbj:toolbarInputField id="Scale"
    value="100%"
    width="50px"
/>
```

using the classlib

```
ToolbarInputField input = new ToolbarInputField("Scale");
input.setType(DataType.STRING);
input.setWidth("50px");
input.setValue("100%");
```

Uniform Resource Name (URN)

Result

The toolbar input field has to be used in the toolbar control. See the "[toolbar \[Page 33\]](#)" control description for the result.

3.4.2.6.33.4 Tool Bar Separator

A separator, usually a horizontal line, that is used to group controls in the toolbar.

- **id**
Identification name of the inputField.
- **transparent**
A boolean value that defines if the design of the toolbar separator. A transparent toolbar separator (enabled = "TRUE") has a different background color.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="sep1" <i>Classlib</i> setId ("sep1")
transparent		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setTransparent (true)

Example**using the taglib**

```
<hbj:toolbarSeparator id="aSeparator"
/>
```

using the classlib

```
ToolbarSeparator sep = new ToolbarSeparator("aSeparator");
set.setTransparent(true);
```

Result

The toolbar input field has to be used in the toolbar control. See the "[toolbar \[Page 33\]](#)" control description for the result.

3.4.2.6.34 Tray

Definition

Similar to [group \[Page 33\]](#) the tray allows grouping of controls. The 'tray' allows additional functionality like collapsing/expanding - similar to the behavior of windows on your Microsoft Windows desktop. The tray control can be for client side eventing. See the [EventValidationComponent \[Page 33\]](#) description for more details.

Portal components (components that will run in the *SAP Enterprise Portal*) are placed in a tray by the portal.

- **design**

The design of the tray can be:

- **BORDER**

The tray has a title bar and the panel has a frame that defines the size.

- **BORDERLESS**

The tray has only a title bar.

- **FORM**

The tray has a title bar. The panel is filled with a background color. The color is different from the title background color.

- **TEXT**

The tray has a title bar. The panel is filled with the same background color as the title bar.


- **enabled** - inherited from [EventValidationComponent \[Page 33\]](#).

A boolean value that enables (=true) or disables (=false) the tray control. A disabled tray sends no event when clicked.

- **id**

Identification name of the tray.

- **isCollapsed**

A boolean value that if "true" shows only the title bar. As indicator that the tray is collapsed, the collapsed symbol is displayed.  When clicking on this symbol the 'onExpand' event is fired.


- **jsObjectNeeded** - inherited from [Component \[Page 33\]](#).

A boolean value that defines if a JavaScript object has to be generated for the tray component.

- **menu**

Set a hover menu for the tray.


- **onCollapse**

Defines the event handling method that will be processed when the user clicks on the collapse symbol .

If the attribute is set to a `<%=null %>` string or the attribute is omitted the symbol is not displayed in the title bar.


- **onEdit** - *Deprecated*

Uniform Resource Name (URN)

Defines the event handling method that will be processed when the user clicks on the collapse symbol .


If the attribute is set to a `<%=null %>` string or the attribute is omitted the symbol is not displayed in the title bar.

- **onExpand**

Defines the event handling method that will be processed when the user clicks on the expand symbol .

This symbol can be active only when 'isCollapsed' is set to "true". If the attribute is set to a `<%=null %>` string or the attribute is omitted the symbol is not displayed in the title bar.

- **onRemove** - Deprecated

Defines the event handling method that will be processed when the user clicks on the expand symbol .

If the attribute is set to a `<%=null %>` string or the attribute is omitted the symbol is not displayed in the title bar.

- **title**

Defines the string of text placed left aligned in the title bar. If no title should be displayed an empty string (null) can be used. The width of the tray is automatically adjusted to the length of the text when the 'width' attribute is set smaller than the title text width.

- **tooltip**

Defines the hint of the tray which is displayed as the mouse cursor passes over the tray, or as the mouse button is pressed but not released.

- **width**

Defines the width of the tray. The width of the button is automatically adjusted to the length of the 'title'. To see an effect of the 'width' attribute 'width' has to be set higher as the width defined thru the length of the 'title' string. If an empty (null) 'title' string is set no 'title' attribute is defined the width of the tray is set according to the 'width' attribute.

Attributes	M	Values	Usage
design		BORDER (d) BORDERLESS FORM TEXT	<i>Taglib</i> design="FORM" <i>Classlib</i> setDesign (TrayDesign.FORM)
enabled*		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEnabled(true)
id	*	String (cs)	<i>Taglib</i> id="Intro_Text" <i>Classlib</i> Object id
isCollapsed		FALSE (d) TRUE	<i>Taglib</i> isCollapsed="TRUE"

Uniform Resource Name (URN)

			<i>Classlib</i> setCollapsed (true)
jsObjectNeeded**		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setJsObjectNeeded(true)
menu		Component	<i>Taglib</i> No tag available <i>Classlib</i> setMenu(HoverMenu hm)
title		String	<i>Taglib</i> title="Headlines" <i>Classlib</i> setTitle("Headlines")
tooltip		String	<i>Taglib</i> tooltip="latest news" <i>Classlib</i> setTooltip("latest news")
width		Unit (50%)	<i>Taglib</i> width="300" <i>Classlib</i> setWidth("300")

Events	M	Values	Usage
onCollapse		String (cs)	<i>Taglib</i> onCollapse="ev_col" <i>Classlib</i> setOnCollapse("ev_col")
onEdit Deprecated		String (cs)	<i>Taglib</i> onEdit="ev_ed" <i>Classlib</i> setOnEdit("ev_ed")
onExpand		String (cs)	<i>Taglib</i> onExpand="ev_ex" <i>Classlib</i> setOnExpand("ev_ex")
onRemove Deprecated		String (cs)	<i>Taglib</i> onRemove="ev_re" <i>Classlib</i> setOnRemove("ev_re")

trayBody

Defines the items in the tray. A tray can be filled with any any control (checkbox, image, textView and so on).

Example

using the taglib

```
<hbj:tray
  id="HeadlineNews"
  design="BORDER"
  title="latest Headlines"
  tooltip="all the news you need"
  onEdit="ev_hd_edit"
  onRemove="ev_hd_rem"
  width="25%"
>

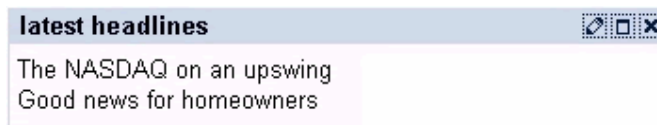
  <hbj:trayBody>
    <hbj:textView
      encode="true"
      text="The NASDAQ on an upswing<br>Good news for homeowners"
    />
  </hbj:trayBody>
</hbj:tray>
```

using the classlib

```
Form form = (Form) this.getForm();
Tray tray = new Tray("HeadlineNews");
tray.setDesign(TrayDesign.BORDER);
tray.setTitle("latest Headlines");
tray.setTooltip("all the news you need");
tray.setOnEdit("ev_hd_edit");
tray.setOnRemove("ev_hd_rem");
tray.setWidth("25%");

TextView ttv2 = new TextView("ttv2");
ttv2.setText("The NASDAQ on an upswing<br>Good news for homeowners");
ttv2.setEncode(false);
tray.addComponent(ttv2);
form.addComponent(tray);
```

Result



3.4.2.6.35 Tree View

Definition

A representation of hierarchical data (for example, directory and file names) as a graphical outline. Clicking expands or collapses elements of the outline. The item in a tree is called `TreeNode`. The nesting depth of `TreeNode`s define the hierarchy level.

The tree has no width attribute. Place the tree in a grid layout control if a certain width is required.

- **id**
Identification name of the tree.
- **offsetForTreeNode**
Defines the distance in pixel used by the control to indent the sub nodes.
- **onTreeClick**
Defines the event handling method that will be processed when the user clicks on the tree. If 'onTreeClick' is specified, the event handling routine is called.
- **rootNode**
Defines the root node of tree. This attribute is used when the tree structure is defined in a bean. The tree node in the bean is created with the command line:

```
TreeNode root = new TreeNode("root", "RootNode");
```
- **rootNodesVisible**
A Boolean value that indicates if the rootNodesVisible.
- **title**
Defines the headline of the tree.
- **tooltip**
Defines the hint of the tree which is displayed as the mouse cursor passes over the tree, or as the mouse button is pressed but not released

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> id="Tree1" <i>Classlib</i> setId ("Tree1")
offsetForTreeNode		Value	<i>Taglib</i> offsetForTreeNode="20" <i>Classlib</i> setOffsetForTreeNode (20)
rootNode	+	String	<i>Taglib</i> rootNode="TreeNode" <i>Classlib</i> setRootNode ("TreeNode")
rootNodesVisible		FALSE TRUE (d)	<i>Taglib</i> rootNodeIsVisible="TRUE"

Uniform Resource Name (URN)

			<i>Classlib</i> setRootNodeIsVisible(true)
title		String	<i>Taglib</i> title="Family tree" <i>Classlib</i> setTitle("Family tree")
tooltip		String	<i>Taglib</i> tooltip="The Addams family" <i>Classlib</i> setTooltip("The Addams family")

+ 'rootNode' is required when the treeNode definition is not made immediately after the tree definition. In this case an error message - indicating that a tree without treeNodes is invalid - is generated.

Events	M	Values	Usage
onTreeClick		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setOnTreeClick("tree_click")

treeNode

Defines the items in the tree. The level of the tree is defined by the nesting depth. A treeNode with sub nodes has an indicator. The indicator is a triangle that shows if the node is expanded or collapsed .

- encode**

A boolean value that defines how the text is interpreted. HTML text formatting commands (for example, <h1>, <i> etc.) can be used to change the display of the text. If there are no formatting commands in the text string, the encode attribute has no effect.

Example:

```
text="<h1><i>Important</i></h1>"
```

encode = "false" Browser output:

Important

the text string is rendered by interpreting the formatting commands.

encode = "true" Browser output: <h1><i>Important</i></h1>

the formatting commands are displayed and not interpreted.

- hoverMenuId**

Defines which hover menu is displayed for this tree node. You can define different trigger methods to display the hover menu. For more details, see hover menu.

Uniform Resource Name (URN)

- **id**

Identification name of the tree.

- **onNodeClick**

Defines the event handling method that will be processed when the user clicks on the text of the node.

- **onNodeClose**

Defines the event handling method that will be processed when the user clicks on the node symbol. The node has to be initially defined in expanded mode (`isOpen=true`) in order to create the event. The event will then always occur when the user clicks on the symbol to expand the node. No event occurs when the tree node expands.

When the `onNodeClose` attribute is set, the tree node does not collapse on the web client. The event is sent to the server and the application has to take care about the further processing (for example, define the sub nodes of the tree node and set the tree node collapsed (`isOpen=false`)).

- **onNodeExpand**

Defines the event handling method that will be processed when the user clicks on the node symbol. The node has to be collapsed initially (`isOpen=false`) in order to create the event. The event will then always occur when the user clicks on the symbol to expand the node. No event occurs when the tree node collapses.

When the `onNodeExpand` attribute is set, the tree node does not expand on the web client. The event is sent to the server and the application has to take care about the further processing (for example, define the sub nodes of the tree node and set the tree node expanded (`isOpen=true`)).



The attributes `onNodeExpand` and `onNodeClose` are useful for trees with a lot of entries (transmission problems possible since the page could become pretty big) or if you want to have full control over the tree nodes and build the sub nodes dynamically. The server application has to take care about the modes of the node itself. If you have set an `onNodeExpand` attribute initially, you have to take care about following steps yourself when the event is fired:

- Create the sub nodes.
- Set the node status (`isOpen=true`).
- Set the `onNodeClose` event to receive an event when the user closes the tree node again.

This works vice versa if you have set an `onNodeClose` attribute initially.

- **open**

A Boolean value that indicates if the node is expanded or collapsed. This attribute only has an effect when the node has at least one sub node. If a node is expanded all sub nodes of the node are displayed. Symbols to indicate the node status:

Node expanded 

Node collapsed 

- **text**

Defines the string of text displayed for the `treeNode`. HTML commands for text formatting (for example, `` for bold characters) can be used..

Uniform Resource Name (URN)

- **tooltip**

Defines the hint of the treeNode which is displayed as the mouse cursor passes over the treeNode, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
encode		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEncode (false)
hoverMenuId		String (cs)	<i>Taglib</i> hoverMenuId="helpHover1" <i>Classlib</i> setHoverMenu (HoverMenu helpHover1)
id	*	String (cs)	<i>Taglib</i> id="TreeNode" <i>Classlib</i> setId ("TreeNode")
open		FALSE TRUE (d)	<i>Taglib</i> isOpen="FALSE" <i>Classlib</i> setOpen (false)
showExpander		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setShowExpander (false)
text		String	<i>Taglib</i> text="Gomez" <i>Classlib</i> setText ("Gomez")
tooltip		String	<i>Taglib</i> tooltip="1 st Family member" <i>Classlib</i> setTooltip ("1 st Family member")

Events	M	Values	Usage
onNodeClick		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setOnNodeClick ("onNodeClick")
onNodeClose		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> setOnNodeClose ("onNodeClose")
onNodeExpand		String (cs)	<i>Taglib</i>

Uniform Resource Name (URN)

			No tag available <i>Classlib</i> <code>setOnNodeExpand("onNodeExpand")</code>
--	--	--	---

Example**using the taglib**

Uniform Resource Name (URN)

```
<hbj:tree
  id="S_Tree"
  title="e-enviroment"
  tooltip="enviroment of my computer"
>
  <hbj:treeNode
    id="e_root"
    text="Desk"
    isOpen="true"
    tooltip="My desk"
  >
    <hbj:treeNode
      id="e_comp"
      text="Computer"
      isOpen="true"
    >
      <hbj:treeNode
        id="e_comp_fl"
        text="Floppy"
      />
      <hbj:treeNode
        id="e_comp_hd"
        text="Harddisk"
      />
      <hbj:treeNode
        id="e_comp_dvd"
        text="DVD"
      />
    </hbj:treeNode>
    <hbj:treeNode
      id="e_net"
      text="Network"
      isOpen="true"
      tooltip="Company network"
    >
      <hbj:treeNode
        id="n_lan"
        text="LAN"
        tooltip="Local Area Network"
      />
      <hbj:treeNode
        id="n_wan"
        text="WAN"
        tooltip="Wide Area Network"
      />
      <hbj:treeNode
        id="n_infra"
        text="Infrared"
        tooltip="Infrared connection"
      />
    </hbj:treeNode>
  </hbj:treeNode>
</hbj:tree>
```

Uniform Resource Name (URN)

using the classlib

```

Form form = (Form) this.getForm();
Tree tree = new Tree("S_Tree", "e-enviroment");
tree.setToolTipText("enviroment of my computer");

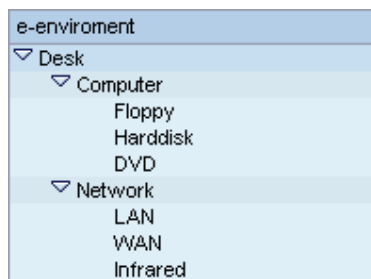
TreeNode root = new TreeNode("e_root", "Desk");
root.setOpen(true);
root.setToolTipText("My desk");

// Tags at the second level -
//      the entries are defined with the event "onName"
// which is fired when the event is clicked.
TreeNode name1 = new TreeNode("e_comp", "Computer", root);
name1.setOnNodeClick("onName");
TreeNode name2 = new TreeNode("e_net", "Network", root);
name2.setOnNodeClick("onName");
TreeNode name11 = new TreeNode("e_comp_fl", "Floppy", name1);
name11.setOnNodeClick("onName");
TreeNode name12 = new TreeNode("e_comp_hd", "Harddisk", name1);
name12.setOnNodeClick("onName");
TreeNode name13 = new TreeNode("e_comp_dvd", "DVD", name1);
name13.setOnNodeClick("onName");
TreeNode name21 = new TreeNode("n_lan", "LAN", name2);
name21.setOnNodeClick("onName");
TreeNode name22 = new TreeNode("n_wan", "WAN", name2);
name22.setOnNodeClick("onName");
TreeNode name23 = new TreeNode("n_infra", "Infrared", name2);
name23.setOnNodeClick("onName");

tree.setRootNode(root);
form.addComponent(tree);

```

Result



Programming Tip

Usually the root node is visible and all sub nodes are displayed on the second level. If you make the root node invisible all sub nodes are displayed on first level.

Uniform Resource Name (URN)

Example

using the taglib

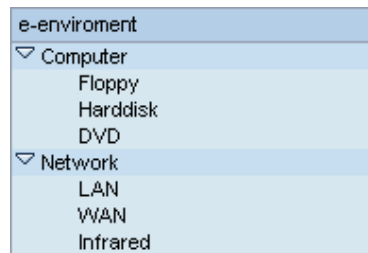
Uniform Resource Name (URN)

```
<hbj:tree
  id="S_Tree"
  title="e-enviroment"
  tooltip="enviroment of my computer"
>
<% S_Tree.setRootNodeIsVisible(false); %>

<hbj:treeNode
  id="e_root"
  text="Desk"
  isOpen="true"
  tooltip="My desk"
>
  <hbj:treeNode
    id="e_comp"
    text="Computer"
    isOpen="true"
  >
    <hbj:treeNode
      id="e_comp_fl"
      text="Floppy"
    />
    <hbj:treeNode
      id="e_comp_hd"
      text="Harddisk"
    />
    <hbj:treeNode
      id="e_comp_dvd"
      text="DVD"
    />
  </hbj:treeNode>

  <hbj:treeNode
    id="e_net"
    text="Network"
    isOpen="true"
    tooltip="Company network"
  >
    <hbj:treeNode
      id="n_lan"
      text="LAN"
      tooltip="Local Area Network"
    />
    <hbj:treeNode
      id="n_wan"
      text="WAN"
      tooltip="Wide Area Network"
    />
    <hbj:treeNode
      id="n_infra"
      text="Infrared"
      tooltip="Infrared connection"
    />
  </hbj:treeNode>
</hbj:treeNode>
</hbj:tree>
```

Result



3.4.2.6.35.1 Usage & Type

The tree view control is used to display hierarchical data or text. The hierarchy levels may be expanded and collapsed. Every tree node contains a text and an arrow icon that expands and collapses the node. If a node has no child elements in the hierarchy, there is no arrow icon. The node text might also link to a function that displays the connected data.

The first four levels have different colors. From the 5th level on the color stays the same like in the 4th level.

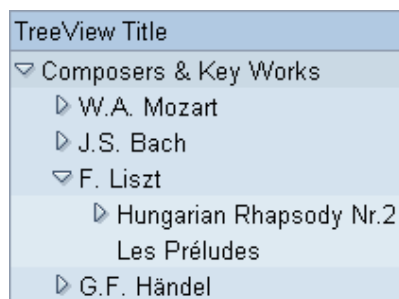


Figure 1: Example of a tree with three levels

Usage

Trees contain complex information and are cumbersome to use. If possible, do not use trees and consider other alternatives, especially in iViews. Trees with hierarchies more than 2-3 levels deep should be avoided altogether!

How to Avoid Trees

If the number of tree elements is small, hierarchies can be flattened to lists, and the items may follow some other ordering like by alphabet or relevance.

Consider using [dropdown list boxes \[Page 33\]](#), the shuffler (filter) or [tabstrips \[Page 33\]](#) in combination with [tables](#) in order to select partial data sets. This leads to a far less complex user interface than large trees that have to be scrolled or paged through.

Design-relevant Attributes

The tree view control does not have a width attribute. To set the width, place the tree inside a grid layout control.

Use attribute title to set a title for the tree.

For tree items, the item text and a corresponding tooltip text can be defined (attributes text and tooltip).

Related Controls

[Item List \[Page 33\]](#), [Dropdown List Box \[Page 33\]](#), [Table View \[Page 33\]](#), [Tabstrip \[Page 33\]](#), [Grid Layout \[Page 33\]](#) (for sizing)

3.4.2.6.35.2 Browser Support & 508

Netscape 4.7 cannot display certain visual nuances of the standard tree control. This tree has no borders and the title height differs from the standard tree (figure 1).

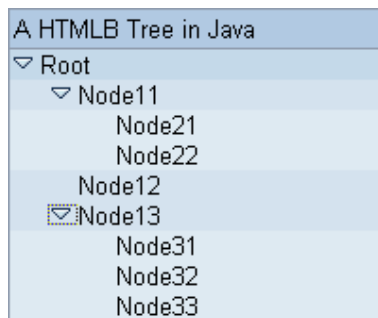


Figure 1: Example of the Standard Tree

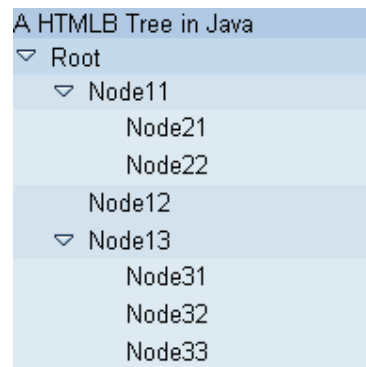


Figure 2: Example of the Tree in Netscape Navigator 4.7

The tree view is always opened completely in Netscape 4.7. It is not possible to expand and collapse nodes locally. The application has to handle these operations (requires server round-trip).

Editability in Style Editor

In the Style Editor, it is possible to modify the following attributes of the tree view control:

Group	Style	IE5 and above	Netscape 4.7
Level Background Colors	Background Color of 1st Level	x	x
	Background Color of 2nd Level	x	x
	Background Color of 3rd Level	x	x
	Background Color of 4th Level	x	
Tree Icons	URL to "Expand" Icon	x	
	URL to "Collapse" Icon	x	

Uniform Resource Name (URN)

	URL to "Node" Icon	x	x
	Height of Tree Icon	x	x
	Width of Tree Icon	x	
Container	Background Color of Container	x	x
	Title	x	x
	Font Color of Container Title	x	
	Height of Container Title	x	
	Container Border	x	
	Bottom Border of Container	x	
	Cell Padding	x	

Table 1: Editable styles for the tree view control

Accessibility - 508 Support

- Keyboard
Each tree node is inserted into the accessibility hierarchy by default.
- Default Description
Is provided by the HTMLB rendering engine for each tree node.
- Application-specific Description
Set an additional description using the setTooltip method for each tree node if needed.

3.4.2.7 Non Visible Controls

Purpose

Non visible controls are also placed in a form, like visible controls, but generate no visible output on the browser.

Non Visible Controls

[Applet Container](#)

[\[Page 33\]Bookmark](#)

[\[Page 33\]Component](#)

[\[Page 33\]Event Validation \(Client Eventing\)](#)

[\[Page 33\]Image Map](#)

[\[Page 33\]Timer \[Page 33\]](#)

3.4.2.7.1 Applet Container

Definition

Provides a container to host an applet.

Uniform Resource Name (URN)

- **classObject**
Specifies the class object of the applet has to be in the library.
- **height**
Defines the height of the applet.
- **library**
Specifies the library that contain the class object and all necessary libraries.
- **name**
Name of the applet. This attribute is passed on to the web client.
- **version**
Version of the applet. This attribute is passed on to the web client.
- **width**
Defines the width of applet.

Attributes	M	Values	Usage
classObject	*	String (cs)	<i>Taglib</i> classObject="appletSample" <i>Classlib</i> setClassObject ("appletSample")
height		Unit	<i>Taglib</i> height="300" <i>Classlib</i> setHeight ("300")
library	*	String (cs)	<i>Taglib</i> library="lib.appletLib.jar" <i>Classlib</i> setLibrary ("lib.appletLib.jar")
name		String	<i>Taglib</i> name="Stock index" <i>Classlib</i> setName ("Stock index")
version			<i>Taglib</i> No tag available <i>Classlib</i> setVersion (JavaVersion.MEDIUM)
width		Unit	<i>Taglib</i> width="450" <i>Classlib</i> setWidth ("450")

Applet Parameter (AppletParameter)

Supplies parameter for the applet.

Uniform Resource Name (URN)

- **name**
Name of the parameter.
- **value**
Value for the 'name' parameter.

Events	M	Values	Usage
name		String (cs)	<i>Taglib</i> name="backgroundColor" <i>Classlib</i> addParameter("backgroundColor", "red")
value		String (cs)	<i>Taglib</i> value="red" <i>Classlib</i> addParameter("backgroundColor", "red")

Example

using the taglib

```
<hbj:appletContainer
  height="300"
  width="400"
  library("lib.appletlib.jar")
  classObject("applet.SampleApplet")>
  <hbj:appletParameter
    name="backgroundColor"
    value="red"
  />
</hbj:appletContainer>
```

using the classlib

```
Form form = (Form) this.getForm();

AppletContainer container = new AppletContainer();
container.setClassObject("com.sap.htmlb.test.applet.SampleApplet");
container.setLibrary("lib.appletlib.jar");
container.setHeight("400");
container.setWidth("400");
container.addParameter("codebase", "/htmlb/applet");
container.addParameter("backgroundColor", "red");

form.addComponent(container);
```

3.4.2.7.2 Bookmark

Definition

A bookmark (also known as "named anchor") is an invisible tag, that marks a certain position in the document.

- **bookmark**

Defines the bookmark. The specified name can used in the link component as 'reference' attribute to return to the bookmark. The 'reference' attribute has to specify a # sign before the bookmark name.

Example - Returning to a bookmark named "chapter1":

'reference' attribute of the `link` component has to be set to: `#chapter1`

Attributes	M	Values	Usage
bookmark		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setBookmark ("chapter1")</code>

3.4.2.7.3 Component

Definition

This is the base class for all components. We describe component because the "jsObjectNeeded" method is necessary for the client side eventing.

- **id**

Identification name of the component.

- **jsObjectNeeded**

A boolean value that defines if a JavaScript object is defined for the component (=true) or not (=false). A JavaScript object is necessary, that you can access the component in your JavaScript program. By default the JavaScript object generation is disabled, to reduce the generated HTML code.

- **parent**

Defines the parent container for the component.

Attributes	M	Values	Usage
id		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <code>setId ("aComponent")</code>
jsObjectNeeded		FALSE (d) TRUE	<i>Taglib</i> <code>jsObjectNeeded="TRUE"</code> <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setJsObjectNeeded(true)</code>
parent		Component	<i>Taglib</i> No tag available <i>Classlib</i> <code>setParent(Container parent)</code>

3.4.2.7.4 Event Validation (Client Eventing)

Definition

This non visible component is necessary for the client side eventing. Client side eventing is a Portal service (See EPCF for more details) that allows event driven manipulations on the web client without causing a server event. The `eventValidationComponent` inherits from the [Component \[Page 33\]](#) component.

Typical client side event tasks are validation of input fields, for example, validating a date or number format. For easier use of the client side event service in combination with HTMLB the client side eventing functions have been integrated into the HTMLB API. All components that support client side eventing (like button, checkBox and inputField) inherit from this component.

- **clientEvent**

Defines the event trigger and the event handler for client side event handling. Following event triggers are possible:

- **ON_BLUR**
The blur event is fired, when a component, for example, an inputField, loses the focus.
- **ON_CHANGE**
The change event is fired, after the ON_BLUR event and if the value of the component had been changed (for example, you changes the value of an inputField).
- **ON_CLICK**
The click event is fired, when a component that can be clicked, like a button or checkbox, has been clicked.
- **ON_FOCUS**
The focus event is fired, when a component, for example, an inputField, gets the focus (for example, by clicking into the inputField).
- **ON_FORM_SUBMIT**
The form submit event is fired, when the submit button has been clicked and before the form is sent to the server. This event gives you the opportunity to validate the form before it is sent to the server.
- **ON_KEYDOWN**
The key down event is fired, when any key, including function keys, on the keyboard is pressed. When you define an ON_KEYPRESS and an ON_KEYDOWN event trigger, the ON_KEYDOWN event handler is called first.

Uniform Resource Name (URN)

When this event handler is finished, the ON_KEYPRESS event handler is called.

- ON_KEYPRESS

The key press event is fired, when a ASCII key, no function key, on the keyboard is pressed. When you define an ON_KEYPRESS and an ON_KEYDOWN event trigger, the ON_KEYDOWN event handler is called first. When this event handler is finished, the ON_KEYPRESS event handler is called.

- ON_KEYUP

The key up event is fired, when the pressed key is released again.

- ON_TIMEOUT

The time out event is fired, when a time out occurs.

- ON_VALIDATION

The validation event is fired, before the validation of the component is called.

The event trigger that can be used, depends on the component. For example, a ON_TIMEOUT event is not possible for a button.

- **enabled**

A boolean value that defines if the component is enabled (= true) or disabled (= false). A disabled component fires no event and usually has a different color.

- **errorText**

Defines the error message that is displayed if the validation is negative.

- **requiresValidation**

A boolean value that defines that the component has to be validated (requiresValidation = true) before a server event is fired.

- **serverEvent**

Defines the event trigger and the event for event handling by the server. Events are for example, BreadCrumbClickEvent, ButtonClickEvent and so on. Refer to the HTMLB Javadoc description - Class "Event" for available HTMLB events.

- **validator**

Defines an application specific validator for a component. If you specify a null value as validator argument, the validator is removed for the component. Validators are for example, CancelButtonValidator, DataTypeValidator, LengthValidator and RequiredValidator. Refer to the HTMLB Javadoc description - Class "Validator" for more details.

Attributes	M	Values	Usage
clientEvent		String (cs)	<i>Taglib</i> No tag available <i>Classlib</i> <pre>setClientEvent(EventTrigger.ON_FOCUS, "onFocus")</pre>
enabled		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i>

Uniform Resource Name (URN)

			<code>setEnabled(true)</code>
errorText		String	<i>Taglib</i> No tag available <i>Classlib</i> <code>setErrorText("invalid date")</code>
requiresValidation		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> <code>setRequiresValidation(true)</code>
serverEvent			<i>Taglib</i> No tag available <i>Classlib</i> <code>setServerEvent (EventTrigger.ON_FOCUS, Event ev)</code>
validator			<i>Taglib</i> No tag available <i>Classlib</i> <code>setValidator (Validator validator)</code>

The following example demonstrates the set up for client side eventing with an [inputField \[Page 33\]](#) component. The "jsObjectNeeded" attribute is inherited from the [Component \[Page 33\]](#) component.

Example

using the taglib

```
<hbj:inputField
  id="currencyDisplay"
  type="BCD"
  width="250px"
  value="100"
  jsObjectNeeded="true">
  currencyDisplay.setClientEvent(EventTrigger.ON_CHANGE,
                                "calculateCurrencyToFrom()");
</hbj:inputField>
```

using the classlib

```
Form form = (Form) this.getForm();

InputField inf2 = new InputField("currencyDisplay");
inf2.setJsObjectNeeded(true);
inf2.setClientEvent(EventTrigger.ON_CHANGE,
                    "calculateCurrencyToFrom()");
inf2.setBCD("100");
inf2.setWidth("250px");

form.addComponent(inf2);
```

3.4.2.7.5 Image Map

Definition

This control contains clickable areas which are associated with an image. The id of the imageMap is associated with the [image \[Page 33\]](#) control (using the 'setImageMap' method). The imageMap control contains imageAreas. The imageAreas contain [links \[Page 33\]](#) which define the links or event handling methods that have to be processed when the imageArea has been clicked.

- **id**

Identification name of the imageMap.

Attributes	M	Values	Usage
id	*	String (cs)	<i>Taglib</i> <code>id="imageMap"</code> <i>Classlib</i> <code>setId("imageMap")</code>

Control API for Image Area (imageArea)

This control is used to define the clickable areas. The areas can be polygons, rectangles and circles. The areas are defined in coordinates. The origin of the coordinates is the upper, left corner of the image.

- **areaType**

Defines the shape of the area:

- CIRCLE

Circular area. The circle is defined by the center and the radius of the circle.

Example (from top=100, from left=200, radius=30): 100,200,30

- POLYGON

The polygon is defined by coordinates of the polygon points.

Example: Triangle (p1=100,100, p2=0,200, p3=200,200):
100,100,0,200,200,100,100

- RECTANGLE

Rectangular area. The rectangle is defined by the upper left and lower right corner.

Example (Upper Left= 100,100, Lower Right=300,200): 100,100,300,200

- **coordinates**

Defines the coordinates of the area according to the areaType. The coordinates are separated by commas.

- **link**

Id of the [link \[Page 33\]](#) control that is associated with the imageArea. The link defines the action (link or event) which is performed when the user clicks on this area.

- **tooltip**

Uniform Resource Name (URN)

Defines the hint of the imageArea which is displayed as the mouse cursor passes over the imageArea, or as the mouse button is pressed but not released.

Attributes	M	Values	Usage
areaType		CIRCLE POLYGON RECTANGLE	<i>Taglib</i> <code>areaType="RECTANGLE"</code> <i>Classlib</i> <code>setAreaType (ImageAreaType.RECTANGLE)</code> (See hint below)
coordinates		String	<i>Taglib</i> <code>coordinates="0,0,100,100"</code> <i>Classlib</i> <code>setCoordinates ("0,0,100,100")</code> (See hint below)
link		String (cs)	<i>Taglib</i> <code>link="idOfLink"</code> <i>Classlib</i> <code>setLink(Link link)</code>
tooltip		String	<i>Taglib</i> <code>tooltip="a image map"</code> <i>Classlib</i> <code>setTooltip("a image map")</code>



When you use the classlib, the areaType and the coordinates have to be specified already when the imageArea is created.

Example:

```
ImageArea circ_left = new ImageArea
(
  com.sap.htmlb.enum.ImageAreaType.CIRCLE,
  "12,165,12");
```

Example

using the taglib

Uniform Resource Name (URN)

```

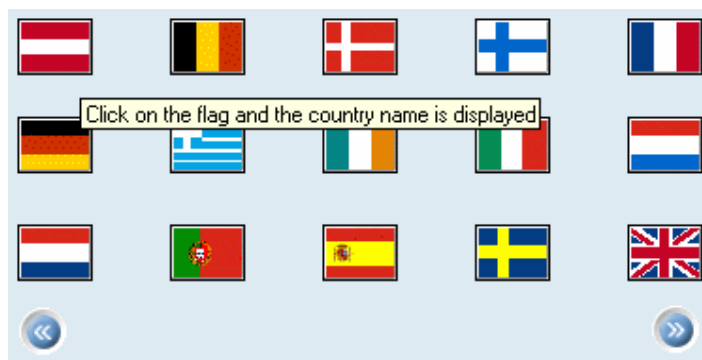
<hbj:link
  id="linkID"
  reference="http://www.sap.com"
/>
<hbj:imageMap
  id="imageMap">
    <hbj:imageArea
      areaType="RECTANGLE"
      coordinates="0,0,39,30"
      link="linkID"
      tooltip="Click here to go to the SAP site"
    />
  </hbj:imageMap>
  <hbj:image
    id="image_logo"
    alt="Image"
    imageMapId="imageMap"
    tooltip="Click on upper/Left area bring you to the SAP site"
    src="">
    <%
      IResource
      rs=componentRequest.getResource(IResource.IMAGE,
        "mimes/EUFlag.gif");
      image_logo.setSrc(rs.getResourceInformation().
        getURL(componentRequest));
    %>
  </hbj:image>

```

using the classlib

see imageMap example in the PDK

Result (from classlib example)



3.4.2.7.6 Timer

Definition

This control fires a server or client event after a preset time. The time is set in 1/1000 seconds. The timer control has no visible effect. The timer control can be for client side eventing. See the [EventValidationComponent](#) description for more details.

- clientEvent**
 Defines the JavaScript fragment that is executed when the time out event occurs. If both events ('serverEventName' and 'clientEvent') are specified, the 'clientEvent' event handling method is activated and the serverEventName is omitted.
- enabled** - inherited from [EventValidationComponent \[Page 33\]](#).
 A boolean value that enables (=true) or disables (=false) the timer control. A disabled timer sends no event when clicked.
- id**
 Identification name of the timer component.
- jsObjectNeeded** - inherited from [Component \[Page 33\]](#).
 A boolean value that defines if a JavaScript object has to be generated for the timer component.
- serverEventName**
 A boolean value that activates (=TRUE) or deactivates (=FALSE) the "Insert Image/Link" function.
- timeOut**
 Time span after which the event is fired. The time value is set in milli seconds.

Attributes	M	Values	Usage
enabled*		FALSE TRUE (d)	<i>Taglib</i> No tag available <i>Classlib</i> setEnabled(true)
id	*	String (cs)	<i>Taglib</i> id="timer" <i>Classlib</i> setId("timer")
jsObjectNeeded**		FALSE (d) TRUE	<i>Taglib</i> No tag available <i>Classlib</i> setJsObjectNeeded(true)
timeout		Value (0)	<i>Taglib</i> timeout="1000" <i>Classlib</i> setTimeout(1000)

* Method is inherited from the [EventValidationComponent \[Page 33\]](#) component.

** Method is inherited from the [Component \[Page 33\]](#) component.

Uniform Resource Name (URN)

Events	M	Values	Usage
clientEvent		with event type	<i>Taglib</i> No tag available <i>Classlib</i> <pre>setClientEvent(EventTrigger.ON_TIMEOUT, " alert('Timeout')")</pre>
clientEvent		String (cs)	<i>Taglib</i> <pre>clientEvent="alert('Timeout')"</pre> <i>Classlib</i> <pre>setClientEvent("alert('Timeout')")</pre>
serverEventName		String (cs)	<i>Taglib</i> <pre>serverEventName="onTimeout"</pre> <i>Classlib</i> <pre>setServerEventName("onTimeout")</pre>

Example

using the taglib

```
<hbj:timer id="timeOut"
  serverEventName="myTimeoutEvent"
  timeOut="12000"
/>
```

using the classlib

```
Form form = (Form) this.getForm();
Timer timer = new Timer("timeOut");
timer.setServerEventName("myTimeoutEvent");
timer.setTimeout(12000);
form.addComponent(timer);
```

3.4.2.8 Models

Purpose

Models describe the data that is displayed by the component, like list box, table view and chart. The model provides methods to add and read items.

3.4.2.8.1 DateNavigatorModel

public interface DateNavigatorModel

Uniform Resource Name (URN)

Constructor detail

```
public DateNavigatorModel()
```

Create DateNavigatorModel.

```
public DateNavigatorModel(IPageContext pc)
```

Create DateNavigatorModel with the page context. Whenever initial settings are made to the DateNavigatorModel (for example, on first display of the datenavigator control the year 1910 should be displayed as centered year and not the actual year) the DateNavigatorModel has to be defined with the page context. Otherwise the initial settings get lost.

The dateNavigator control is displayed according to the locale settings of the portal or the web client. For more information about locale settings of the portal and the web client, see [documentation regarding Internationalization](#)".

Base data model for the dateNavigator.

Method	Description	Argument	Return value
getCalendar	Returns the calendar instance used by the DateNavigatorModel.	()	java.util.Calendar
getCenteredMonth	Returns the month that is displayed in the center position of all the month displayed.	()	java.lang.Integer
getCenteredYear	Returns the calendar year of the month displayed in the center position.	()	java.lang.Integer
getLocale	Returns the locale (country) code. It takes the locale from the IPageContext.	()	java.util.Locale
getSelectedDays	Returns the the DayRange instance.	()	DayRanges
getToday	Returns the date that should be rendered as the current date.	()	java.util.Date
isLocaleUnknown	Returns a boolean value indicating if the locale has been set.	()	Boolean
setCenteredMonth	Sets the month that should be displayed in the center position of all the month displayed.	(int centerMo)	void
setCenteredYear	Sets the calendar year of the month displayed in the center position.	(int centerYr)	void

Uniform Resource Name (URN)

setLocale	Sets the locale (country) code that should be used for date calculation and presentation.	(java.util.Locale l)	void
setLocaleUnknown	If set to true the locale setting of the browser is used for date calculation and presentation.	(boolean locale)	void
setSelectedDays	Sets the DayRanges instance.	(DayRanges selDay)	void
setToday	Sets the day that should be displayed as the current date.	(java.util.Date today)	void

class DayRanges

The DayRanges class specifies any number of days that should be displayed in the selected state. Every day can have a tooltip.



The month values used in DayRanges correspond with java.util.Calendar. According to that the months start with 0 (for January).

Method	Description	Argument	Return value
add	Adds a date to this instance of DayRanges.	(java.util.Date day)	void
add	Adds a date range to this instance of DayRanges.	(java.util.Date begin, java.util.Date end)	void
add	Adds a date range to this instance of DayRanges that will be displayed as selected.	(java.util.Date begin, java.util.Date end, int selection)	void
add	Adds a date range with a tooltip to this instance of DayRanges.	(java.util.Date begin, java.util.Date end, java.lang.String tooltip)	void
add	Adds a date range with a tooltip to this instance of DayRanges that will be displayed as selected.	(java.util.Date begin, java.util.Date end, java.lang.String tooltip, int selection)	void
add	Adds a date to this instance of DayRanges that will be displayed as selected.	(java.util.Date day, int selection)	void
add	Adds a date to this instance of DayRanges that will be displayed as selected.	(java.util.Date day, java.lang.String tooltip)	void

Uniform Resource Name (URN)

add	Adds a date with a tooltip to this instance of DayRanges that will be displayed as selected.	(java.util.Date day, java.lang.String tooltip, int selection)	void
addMonth	Adds all days of the specified month to this instance of DayRanges	(int month, int year)	void
addMonth	Adds all days of the specified month to this instance of DayRanges that will be displayed as selected.	(int month, int year, int selection)	void
addMonth	Adds all days of the specified month with a tooltip to this instance of DayRanges.	(int month, int year, java.lang.String tooltip)	void
addMonth	Adds all days of the specified month with a tooltip to this instance of DayRanges that will be displayed as selected..	(int month, int year, java.lang.String tooltip, int selection)	void
addWeek	Adds all days of the specified week to this instance of DayRanges.	(int week, int year)	void
addWeek	Adds all days of the specified week to this instance of DayRanges that will be displayed as selected.	(int week, int year, int selection)	void
addWeek	Adds all days of the specified week with a tooltip to this instance of DayRanges.	(int week, int year, java.lang.String tooltip)	void
addWeek	Adds all days of the specified week with a tooltip to this instance of DayRanges that will be displayed as selected.	(int week, int year, java.lang.String tooltip, int selection)	void
deleteTimeInfo	Deletes the time information in the specified calendar.	(java.util.Calendar cal)	void
getCalendar	Returns the Calendar instance associated with this DayRanges instance.	()	java.util.Calendar

Uniform Resource Name (URN)

getDate	Returns a <code>java.util.Date</code> construct out of the specified day, month and year.	(int day, int month, int year)	<code>java.util.Date</code>
getSelectionMode	Returns the selection mode for the specified date.	(<code>java.util.Date</code> date)	int
getTooltip	Returns the tooltip for the specified date.	(<code>java.util.Date</code> date)	<code>java.lang.String</code>
isInRange	Returns a boolean value indicating if the specified date has been added to this instance of <code>DayRanges</code> .	(<code>java.util.Date</code> date)	boolean
isInRange	Returns a boolean value indicating if the specified date has been added to this instance of <code>DayRanges</code> .	(int day, int month, int year)	boolean
isLocaleUnknown	Returns a boolean value indicating if the locale has been set.	()	boolean
normalizeDate	Method to delete all time info in a <code>Date</code> . The given calendar is used to safely convert the given <code>Date</code> to a <code>Date</code> with the same day/month/year but with all time fields zeroed. The calendar instance will not be modified by this method (for example, the inherent time info inside the calendar will be preserved).	(<code>java.util.Calendar</code> cal, <code>java.util.Date</code> date)	<code>java.util.Date</code>
setLocale	Sets the locale (country) code that should be used for date calculation and presentation.	(<code>java.util.Locale</code> l)	void
setLocaleUnknown	If set to <code>true</code> the locale setting of the browser is used for date calculation and presentation.	(boolean localeUnknown)	void

Related Topics

[Using beans and models \[Page 33\]](#)

3.4.2.8.2 IChartModel

public interface IChartModel

Base data model for the chart control.

Implementing classes: JCOChartModel, VectorChartModel, DefaultChartModel

IChartModel and DefaultChartModel

Method	Description	Argument	Return value
firstRow	Sets the pointer of the actual row to first row in the model and returns the number of the first row.	()	int
getColor	Returns the color value of the current row.	()	java.lang.String
getExtension	Returns the extension value of the current row. The extension is a text string that is displayed as tooltip when you move the mouse cursor over the chart element. It shows effect on bar and pie charts, but no effect on line charts.	()	java.lang.String
getFieldName	Returns the field name of the actual row and the specified column.	(int col)	java.lang.String
getFieldValue	Returns the field value of the actual row and the specified column.	(int col)	java.lang.String
getGroupId	Returns the group id of the actual row.	()	java.lang.String
getNumColumns	Returns the number of columns in the model.	()	int
getNumRows	Returns the number of rows in the model.	()	int
getRow	Returns the row position of the pointer (actual row).	()	int
getX	Returns the X value of the actual row.	()	java.lang.String
getY	Returns the Y value of the actual row.	()	java.lang.String
nextRow	Sets the pointer to the next row in the model. If the point has reached the end of the model the method returns false.	()	boolean

JCOChartModel (Methods in addition to the IChartModel/DefaultChartModel)

Method	Description	Argument
--------	-------------	----------

Uniform Resource Name (URN)

setDataSrc	Sets a JCO data table into this model.	(com.sap.mw.jco.JCO. Table DataSrc)
------------	--	--

VectorChartModel (Methods in addition to the IChartModel/DefaultChartModel)

Method	Description	Argument
addItem	<p>Adds an entry with extension to the model.</p> <p>The extension is a text string that is displayed as tooltip when you move the mouse cursor over the chart element. It shows effect on bar and pie charts, but no effect on line charts.</p> <p>If you use a pie chart to display the data, the X values will be used.</p>	(java.lang.String groupid, java.lang.String x, java.lang.String y, java.lang.String color, java.lang.String extension)
addItem	<p>Adds an entry with an URL and an alternate text to the model.</p> <p>The alternate text is a text string that is displayed as tooltip when you move the mouse cursor over the chart element.</p> <p>URL and alternate text show effect on bar and pie charts, but no effect on line charts.</p> <p>If you use a pie chart to display the data, the X values will be used.</p>	(java.lang.String groupid, java.lang.String x, java.lang.String y, java.lang.String color, java.lang.String url, java.lang.String altText)

Example**Setting up the VectorChartModel**

```
// create model
VectorChartModel model = new VectorChartModel();
// add two items of the same group with color 20 and an extension

model.addItem("gr1", "A", "7", "20", "Information to block A");

model.addItem("gr1", "B", "4", "20", "Information to block B");
// add two items of the same group with color 10, an URL and an
// alternate text

model.addItem("gr2", "10", "3", "10", "http://www.sap.de", "Link to SAP");
model.addItem("gr2", "20", "1", "10", "http://www.bmw.de", "Link to BMW");
```

Setting up the JCOChartModel

Uniform Resource Name (URN)

```
// create a new JCO table
JCO.Table table = new JCO.Table("DAX");

// add the info/header to the table. The header is defined by the
// column name, data type and length (see JCO table API for details).
table.addInfo("GROUPID", JCO.TYPE_CHAR, 50);
table.addInfo("X", JCO.TYPE_CHAR, 50);
table.addInfo("Y", JCO.TYPE_CHAR, 50);
table.addInfo("Z", JCO.TYPE_CHAR, 50);
table.addInfo("COLOR", JCO.TYPE_CHAR, 50);
table.addInfo("EXTENSION", JCO.TYPE_CHAR, 150);

// append a record to the table. setValue sets the value as string.
// The second parameter is the column. The numbers are according to
// the sequence of the definition of the header (see above).
table.appendRow();
table.setValue("07.2001", 0);
table.setValue("SAP", 1);
table.setValue("158", 2);
table.setValue("20", 3);
table.setValue("10", 4);
table.setValue("href=\"http://www.sap-ag.de/\"", 5);

..
/ create a JCOChartModel and set the JCO table
model = new JCOChartModel();
((JCOChartModel) model).setDataSrc(table);
```

Related Topics

[Using beans and models \[Page 33\]](#)

3.4.2.8.3 IListModel

Definition

public interface IListModel

Base data model for breadCrumb, dropdownListBox and listBox.

Structure

Method	Description	Argument	Return value
addItem	Add item (entry) to the list	(java.lang.String key, java.lang.String text)	void
addSelection	Selects an item with the given key. Clicking on an selected item does NOT fire an event.	(java.lang.String sel)	void

Uniform Resource Name (URN)

getKeyByIndex	Returns the key of an item with specified index.	(int index)	java.lang.String
getKeys	Returns an iterator for all keys in the list.	()	java.util.Iterator
getMultiSelection	Returns the keys of the currently selected items. The model must be a multi selection model.	()	lang.String[]
getNameOfKeyColumn	Returns the name of the column containing the key values if the list model is associated with a table (for example, JCListModel).	()	java.lang.String
getNameOfTextColumn	Gets the name of the column containing the visible texts if the list model is associated with a table (for example, JCListModel).	()	java.lang.String
getSingleSelection	Gets the key of the currently selected item if the model is a single selection model.	()	java.lang.String
getTextByIndex	Returns the visible text of an item with the specified index.	(int index)	java.lang.String
getTextForKey	Returns the text associated with the key.	(java.lang.String key)	java.lang.String
isSelected	Returns the status (selected or not selected) for an item with the specified index.	(int index)	boolean
isSelected	Returns the status (selected or not selected) for an item with the specified index.	(java.lang.String key)	boolean
isSingleSelection	Returns the status of the model (Singleselect or Multiselect).	()	boolean
removeSelection	Deselects an item with the given key.	(java.lang.String sel)	void
setNameOfKeyColumn	Sets the name of the column containing the key values if the list model is associated with a table (for example, JCListModel). This method has now visible effect.	(java.lang.String n)	void
setNameOfTextColumn	Sets the name of the column containing the visible texts if the list model is associated with a table (for example, JCListModel). This method has now visible	(java.lang.String n)	void

Uniform Resource Name (URN)

	effect.		
setSelection	Selects an item with the given key. The selection from previous selected items is removed. Clicking on an selected item does NOT fire an event.	(java.lang.String key)	void
setSingleSelection	Defines the selection mode for the model. Set to true one item can be selected at one time, set to false multiple entries can be selected. If the model is set to multiple select mode, no event for the control must be defined to avoid events when multiple selects are made. Multiple selects can be made by holding the <Shift> or <Strg>/<Ctrl> key down and click on the entries.	(boolean selection)	void
size	Returns the number of items/entries in the list.	()	int

Example**Setting the model**

Uniform Resource Name (URN)

```
import com.sap.htmlb.BreadCrumb;
import com.sap.htmlb.IListModel;
import com.sap.htmlb.DefaultListModel;

public class BreadCrumbBean {
    private IListModel model;

    /* Constructor - set the basic item (Home) */

    public BreadCrumbBean() {

        /* create model */
        model = new DefaultListModel();
        /* add items to the model */
        model.addItem("start", "Start");
        model.addItem("level1", "1stVisitedPage");
        model.addItem("level2", "2ndVisitedPage");
        model.addItem("actlevel", "ActualLevel");
        /* pre select an item in the model */
        model.addSelection("level1");
    }

    /* get and set method for the model */

    public IListModel getModel() {

        return this.model;
    }

    public void setModel(IListModel bc) {

        model = bc;
    }
}
```

Getting selected items

The example assumes that a listBox in a JSP has the attribute 'onSelect' set, so that an event is fired when the user clicks in the list. We define a method 'onSelect' in the DynPage to handle the event. The method grabs the listBox id, retrieves the selected items and put the string of the selected items in a StringBuffer.

Uniform Resource Name (URN)

```
public void onSelect(Event event) throws PageException {
    /* Handles event from the listbox
       get the listbox by name. The listbox id in the JSP is "LB_Pick" */
    ListBox lb = (ListBox) this.getComponentByName("LB_Pick");

    /* retrieve the items with method "getMultiSelection */
    String[] selection = lb.getMultiSelection();
    StringBuffer sel = new StringBuffer();

    /* get number of items in selection with the "length" method
       - for loop to store it in StringBuffer */
    for (int i = 0; i < selection.length; i++) {
        sel.append(selection[i] + " ");
    }
}
```

Related Topics

[Using beans and models \[Page 33\]](#)

3.4.2.8.4 TableViewModel

public interface TableViewModel

All known implementing classes: DefaultTableViewModel, JCOTableViewModel

The TableViewModel interface specifies the methods the TableView will use to interrogate a tabular data model. The tableView can be set up to display any data model which implements the TableViewModel interface.

The TableView model uses vectors to supply the tableView with data. Visible columns are a subset of the data in your model.

data in model - "invisible" columns					
tableView					
	PLVAR	LOCTP	LOCID	LOCSH	LOCTX
<input type="checkbox"/>	01	F	50000484	Walldorf	training center Walldorf
<input type="checkbox"/>	01	F	50000734	Zurich	
<input type="checkbox"/>	01	F	data in model - "visible" columns		
<input type="checkbox"/>	01	F	50000734		training center Los Angeles
<input type="checkbox"/>	01	F	50000734	Philadelphia	training center Philadelphia
<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>					4 / 16
data in model - "invisible" columns					

The absolute position in the data source has to be calculated by the application.

Method	Description	Argument	Return value
addColumn	Adds a column with the specified name to the visible columns.	(java.lang.String colNam)	TableColumn
addKeyColumn	Adds a key column.	(int columnIndex)	void
getColumnAt	Returns the column for the specified index.	(int columnIndex)	TableColumn
getColumnCount	Returns the number of columns in the model.	()	int
getColumnName	Returns the name of the column for the specified index.	(int columnIndex)	java.lang.String
getColumns	Returns a vector to the visible columns.	()	java.util.Vector
getKeyColumn	Returns the key column.	()	IndexedLinkedList
getRowCount	Returns the number of rows in the model.	()	int
getValueAt	Returns the value of the cell for the specified column and row index.	(int rowInd, int collnd)	AbstractDataType [Page 33]

Uniform Resource Name (URN)

getValueAt	Returns the value of the cell for the specified column and row index.	(int rowInd, java.lang.String colKey)	AbstractDataType [Page 33]
removeColumn	Removes a visible column. The data source is not effected.	(java.lang.String columnName)	void
setColumnName	Sets the name of the column for the specified columnIndex to the specified columnName.	(java.lang.String colNam, int columnIndex)	void
setKeyColumn	Sets the key column for the model.	(int columnIndex)	void
setValueAt	Sets the value for the specified location.	(AbstractDataType aVal, int rowInd, int columnInd)	void

Example

```
// Create a new table model with three columns and add data:
private DefaultTableViewModel createNewTable
    (DefaultTableViewModel model) {
    Vector data = createData();
    Vector colName = new Vector();
    /* Define column names */
    colName.addElement("1stColumn");
    colName.addElement("2ndColumn");
    colName.addElement("3rdColumn");
    model = new DefaultTableViewModel(data, colName);
    return model;
}

private Vector createData() {
    Vector dataVec = new Vector();
    Vector retVector = new Vector();

    /* 1st entry */
    dataVec.addElement("Row 1, Column 1");
    dataVec.addElement("Row 1, Column 2");
    dataVec.addElement("Row 1, Column 3");
    retVector.addElement(dataVec);

    /* 2nd entry */
    dataVec = new Vector();
    dataVec.addElement("Row 2, Column 1");
    dataVec.addElement("Row 2, Column 2");
    dataVec.addElement("Row 2, Column 3");
    retVector.addElement(dataVec);

    /* more entries */
    .
    .
    return retVector;
}
```

Additional documentation on tableView:

[How to setup and work with the onCellClick event \[Page 33\]](#)

[How to setup and work with the onHeaderClick event \[Page 33\]](#)

[How to setup and work with the onNavigate event \[Page 33\]](#)

[How to setup and work with the onRowSelection event \[Page 33\]](#)

Related Topics

[Using beans and models \[Page 33\]](#)

3.4.2.8.4.1 AbstractDataType

public class AbstractDataType

Base class for different supported data types

Method	Description	Argument	Return value
dataTypes	Returns an iterator of all known DataTypes that can be handled.	()	java.util.Iterator
getInstance	Returns an instance a the specified data type.	(DataType type)	AbstractDataType
getType	Returns the instance of the data type enum class that identifies the data type.	()	DataType
isLocaleSpecific	Determines if a data type is locale specific, therefore valued need formatting before output.	()	boolean
isValid	Checks if the user input is valid for this type and returns false if it is not valid. If the user input was invalid, method datatype.getValueAsString() can be used to return the value as string without checks.	()	Boolean
setValue	Sets the value of the data type.	(java.lang.Object v)	void
setValue	Sets the value of the data type.	(java.lang.Object v, java.util.Locale l)	void
toString	Returns the data type value represented as string.	(IPageContext pc)	java.lang.String
toString	Returns the data type value	(java.util.Locale l)	java.lang.String

	represented as string.		
--	------------------------	--	--

3.4.2.9 Beans

Purpose

The bean concept - reusable components that can be used in more than one software package - plays an important role in the portal component development. Portal components can use the beans to store and retrieve data. The JSP as well as the servlet have read and write access to the bean so the bean can be used to transfer data between the JSP and the servlet. In the portal a servlet can be a DynPage or an Abstract Portal Component.

Topics

[Data Storing & Retrieving in the Portal \[Page 33\]](#)

[Usage & Scopes \[Page 33\]](#)

[Using a Bean with the Taglib \[Page 33\]](#)

3.4.2.9.1 Data Storing & Retrieving in the Portal

The methods to store and retrieve data in the portal differ in the lifetime and accessibility. Lifetime is hard to guarantee in the portal. Lifetime can be very short - one request - or last the whole session. When using session keep in mind that the lifetime of a session is controlled by "external" events, like low resources on the server or time-out of the session, which will destroy the stored data in a session. The accessibility controls how "visible" the stored data is - can it be accessed by all users of the same component or only one user.

Storing data in the portal is generally not persistent. If you need persistent storing is needed we recommend a database.

In the Portal we have:

- **IPortalComponentContext**
The IPortalComponentContext defines a user specific view on a Portal Component.
- **IPortalComponentSession**
The IPortalComponentSession is the portal component's view on the servlet session. All objects stored in the Portal Component Session will be stored exclusively for the triple consisting of the Portal Component, the user id and the assigned scope. The lifetime of the shared session that is maintained by the portal environment should be determined by the portal setup only!
The IPortalComponentSession is a "sub session" of `request.getServletRequest().getSession()`. It is unique per user and Portal Component.
- **ServletRequest**
ServletRequest is global for all Portal Components rendered in the same request (equals one Portal Object Model (POM) tree).

Uniform Resource Name (URN)



Be aware that even Portal Components on the same page are rendered in EP in different requests, because they are rendered in separate iFrames on the page! So you can't use the Servlet Request to transfer data between Portal Components! To use the ServletRequest makes only sense, if you combine two portal components in one request using the portal object model.

- Node

Node is a unique element in the Portal Object Model (POM) tree.

- HTTP Session

HTTP Session is global for all Portal Components in the same HTTP Session. You can exchange via HTTP Session data even on different pages.

The following table give you an overview of the different methods to store and retrieve data and the accessibility, persistence and lifetime. The table uses the variables:

```
IPortalComponentRequest: request
IPortalComponentSession: session = request.getComponentSession()
IPortalComponentContext: context = request.getComponentContext()
IPortalComponentProfile: profile = context.getProfile()
```

The table headers use following abbreviations:

S = Session specific

C = Component specific

N = Node specific

U = User specific

P = Persistent

Storing in/Methods	S	C	N	U	P	Lifetime
IPortalComponentContext get: <code>context.getValue()</code> put: <code>context.putValue()</code>		✓		✓		Refresh of component context
IPortalComponentProfile Deprecated get: <code>profile.getValue()</code> put: <code>profile.putValue()</code> scope for beans: <code>application</code>		✓		✓		
IPortalComponentProfile get: <code>pro.getPropertyAttribute("key", "val")</code> put: <code>prof.setProperty("key", "val")</code>		✓		✓	✓	

Uniform Resource Name (URN)

IPortalComponentSession <code>get: session.getValue()</code> <code>put: session.putValue()</code> Option: Declaration of a scope <code>public static short SCOPE_UNIQUE = 0;</code> <code>public static short SCOPE_CONTEXT = 1;</code> <code>public static short SCOPE_COMPONENT = 2;</code> Default: SCOPE_UNIQUE Scope for beans: session	✓	✓	✓ *	✓		Maximum as long as the HTTP session
Servlet Request <code>get: request.getServletRequest().getAttribute()</code> <code>put: request.getServletRequest().setAttribute()</code> scope for beans: request	✓			✓		Request
Node <code>get: request.getNode().getValue()</code> <code>put: request.getNode().putValue()</code>	✓	✓	✓	✓		
HTTP Session <code>get: session.getHttpSession().getValue()</code> <code>put: session.getHttpSession().putValue()</code>	✓			✓		HTTP session

* Only if scope is set to SCOPE_UNIQUE

3.4.2.9.2 Usage & Scopes

The bean concept - reusable components that can be used in more than one software package - plays an important role in the portal component development. Portal components can use the beans to store and retrieve data. The JSP as well as the servlet have read and write access to the bean so the bean can be used to transfer data between the JSP and the servlet. In the portal a servlet can be a DynPage or an Abstract Portal Component.

Declaration of a bean in the Servlet

In the servlet the bean is usually defined with a name, a constructor and set and get methods to handle the data. Some HTMLB-Controls need specific models. In the following example we take the IListModel which is used by the listBox, dropdownListBox and the breadCrumb control (other controls using models would be the dateNavigator, chart, tableView and tree). To have access to the model in the JSP, the get and set method in the servlet has to follow the naming convention:

Uniform Resource Name (URN)

```
get+MethodName
```

The `MethodName` must start with a capital (upper case) letter.

Example

Declaration of the bean with a model

```
public class ListBoxBean {
    private IListModel ListBoxmodel;
    // get model method
    public IListModel getMyListBoxmodel() {
        return this.ListBoxmodel;
    }
    .
    .
}
```

Using the bean in the JSP with a listBox

```
.
.
<!-- Declare Bean --%>
<jsp:useBean id="myBean" scope="application" class="bean.ListBoxBean" />
<!-- Use Bean in model attribute. We use the id myBean (from the
      useBean statement above) and the name of the get method without
      the get --%>

<hbj:listBox
    id="LB_Pick"
    width="100"
    size="5"
    selection="1"
    model="myBean.myListBoxmodel"
/>
.
.
```

Declaration of a Bean in the JSP

The bean has to be declared at the beginning of the JSP.

```
<!-- Declare Bean --%>
<jsp:useBean id="myBean" scope="application" class="bean.ListBoxBean" />
```

Attributes

Attribute	M	Values	Description	Usage
class	*	String (cs)	Defines the class name of the bean.	<code>class="bean.myBean"</code>

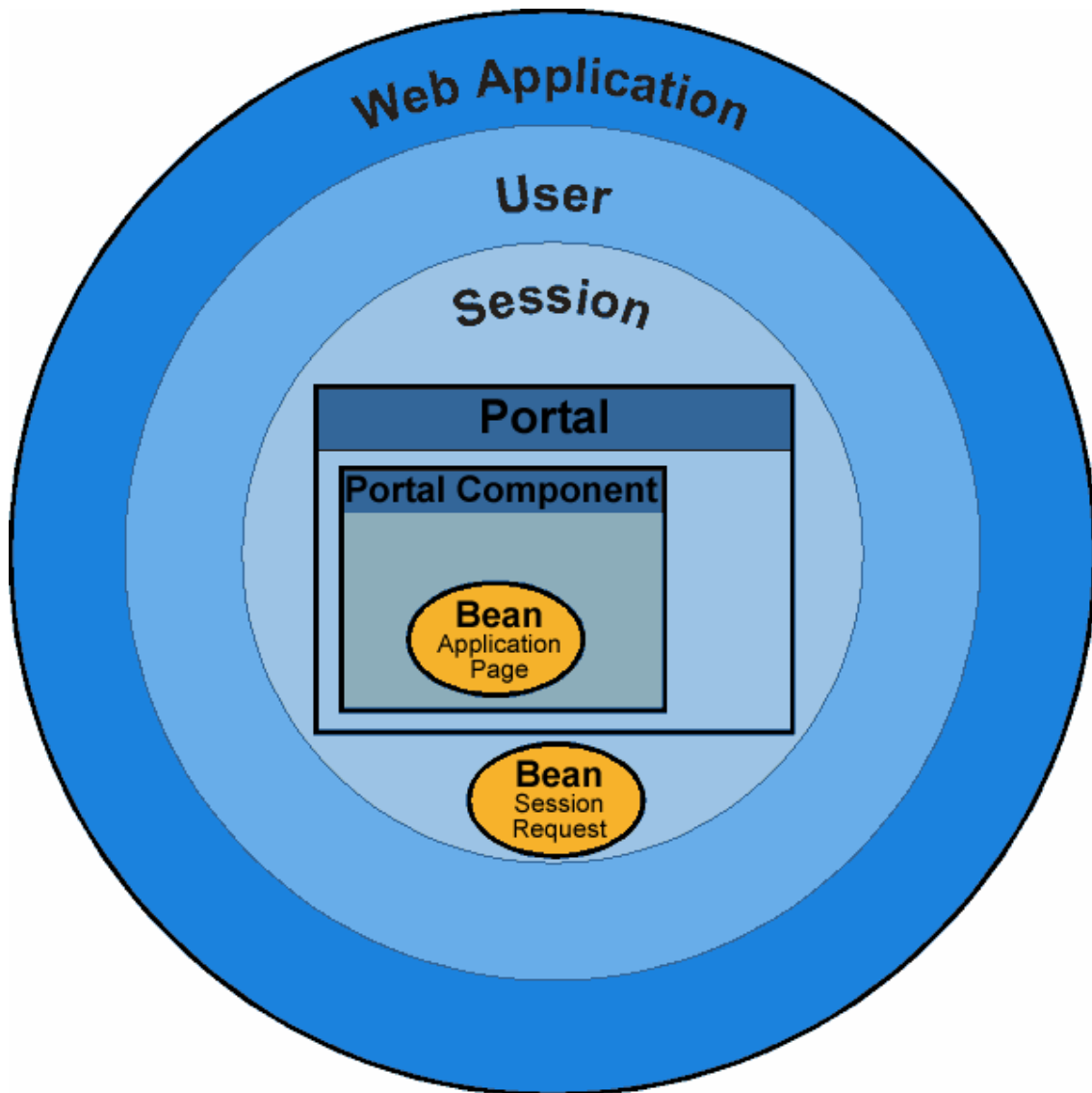
Uniform Resource Name (URN)

id	*	String (cs)	Id of the bean in the JSP. The id references the bean in the JSP.	id = "myBean"
scope	*	APPLICATION SESSION REQUEST PAGE	Defines the scope in which the bean can be accessed.	scope = "application"

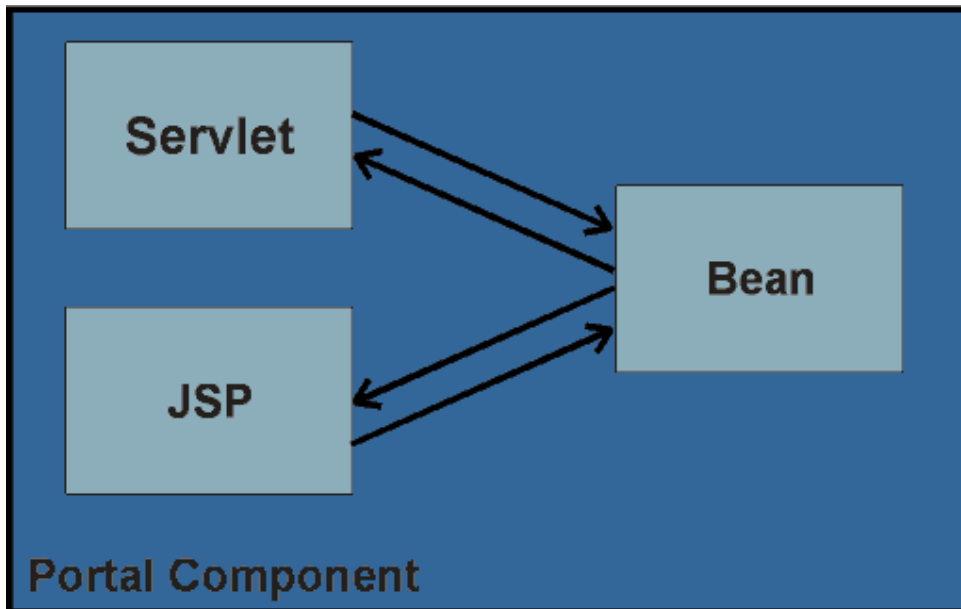
Scope in detail

Except the scope option APPLICATION all the other scope options follow the JSP specifications from Sun Microsystems. The option APPLICATION had to be modified to meet the requirements for a portal. The standard recommendation of APPLICATION would allow access to a the bean through out the whole portal (it would be located in the "Web Application" shell if you look at the following overview chart). In the portal the sphere for APPLICATION is defined as the portal component. This gives the portal component control over the bean but the bean cannot be accessed by other users or other applications of the same user.

Overview



The overview chart shows the location of the bean in the portal according to the scope attribute. The scope attribute also controls the access of servlet and JSP to the bean.

Scope = Application

The bean is "inside" the portal component. JSP and servlet have read and write access to the bean.

Accessing the bean

Get a value:

```
Object value =
    request.getComponentContext().getProfile().getValue(String key);
```

Put/set a value:

```
request.getComponentContext().getProfile().putValue
    (String key, Object value);
```

Scope = Session

The bean is "outside" the portal component. JSP and servlet have read and write access to the bean.

Accessing the bean

Uniform Resource Name (URN)

HTTPServlet:

Get a value:

```
Object value = request.getSession().getValue(String key);
```

Put/set a value:

```
request.getSession().putValue(String key, Object value);
```

Portal:

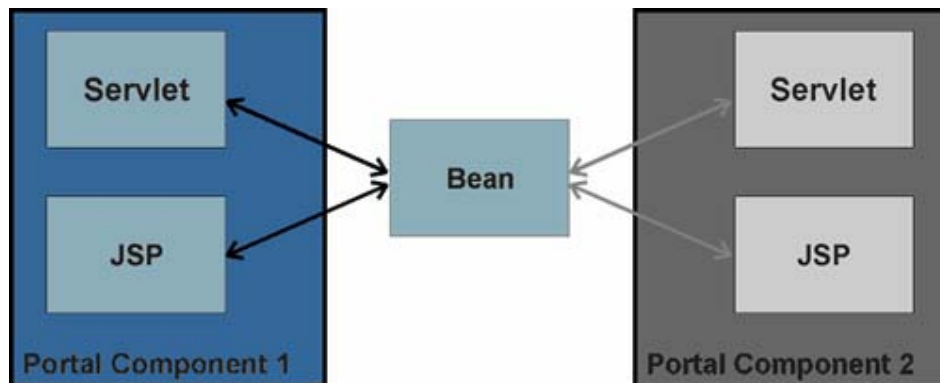
Get a value:

```
Object value = componentRequest.getComponentSession().getValue  
                                                    (String key);
```

Put/set a value:

```
componentRequest.getComponentSession().putValue  
                                                    (String key, Object value);
```

Scope = Request



The bean is "outside" the portal component. JSP and servlet have read and write access to the bean. Because of the location of the bean other portal components in the same request can access the bean as well.



Be aware that even Portal Components on the same page are rendered in EP in different requests, because they are rendered in separate iFrames on the page! So you can't use in general the Servlet Request to transfer data between Portal Components! To use the ServletRequest makes only sense, if you combine two portal components in one request using the portal object model.

This scope requires a careful selection of the bean name.

Uniform Resource Name (URN)

Portal Component 1: Uses bean "myBean" with getName and setName methods.

Portal Component 2: Uses bean "myBean" with getNumber and setNumber methods.

Portal Component 1 is loaded first, then Portal Component 2 is loaded.

Result:

This would cause access error message, when Portal Component 2 uses the getNumber or setNumber; the bean "myBean" with getName and setName is in charge, because Portal Component 1 has been loaded first.

Accessing the bean

HTTPServlet:

Get a value:

```
Object value = request.getAttribute (String key);
```

Put/set a value:

```
request.setAttribute (String key, Object value);
```

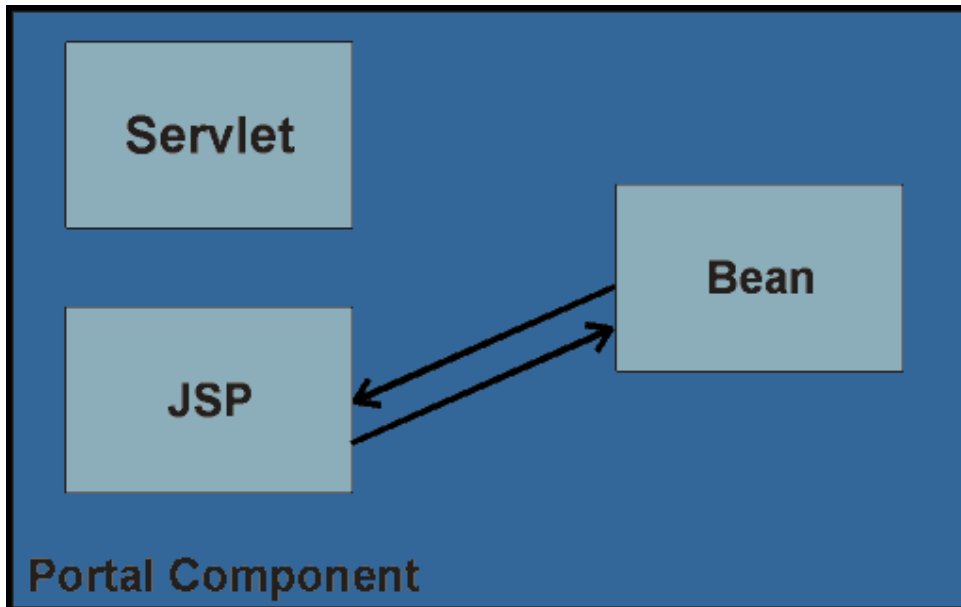
Portal:

Get a value:

```
Object value = componentRequest.getServletRequest().getAttribute  
                                     (String key);
```

Put/set a value:

```
componentRequest.getServletRequest().setAttribute  
                                     (String key, Object value);
```

Scope = Page

The bean is "inside" the portal component. Only the JSP has read and write access to the bean.

Accessing the bean

Get a value:

```
Object value = pagecontext.getValue (String key);
```

Put/set a value:

```
pagecontext.putValue (String key, Object value);
```

3.4.2.9.3 Using a Bean with the Taglib

The bean has to be declared at the beginning of the JSP.

```
<jsp:useBean id="myBean" scope="application" class="bean.myBean" />
```

See [Beans: Usage & Scopes \[Page 33\]](#) for details about the attributes.

To specify the model for the control with the taglib without a scriptlet, a property section in the bean must be defined. In addition the bean needs set and get methods to have access to the bean. The get and set method for the bean has to follow the naming convention:

```
get+MethodName
```

The `MethodName` must start with a capital (upper case) letter.

Uniform Resource Name (URN)

The following example shows the definition of the bean and then the specification in the JSP. The get and set methods in the bean are called `setModel` and `getModel`. The JSP accesses the bean with the bean id and the method `model`.

Example

Definition of the bean:

```
package bean;

import com.sap.htmlb.BreadCrumb;
import com.sap.htmlb.IListModel;
import com.sap.htmlb.DefaultListModel;

public class BreadCrumbBean {

    /* Property declaration */
    private IListModel model;

    /* Constructor */
    public BreadCrumbBean() {
        model = new DefaultListModel();
        model.addItem("home", "Home");
    }

    /* Defining methods */
    public IListModel getMyDefaultListModel() {
        return this.model;
    }
    public void setMyDefaultListModel(IListModel bc) {
        model = bc;
    }
}
```

Using the bean in the JSP

```
<!-- We introduce the bean to the JSP with the id "myBean". --%>
<!-- With the id we have access to the bean in the JSP. --%>
<!-- As class name we have to specify the package name --%>
<!-- plus the class name of the bean - --%>
<!-- that is "bean.BreadCrumbBean" --%>
<jsp:useBean id="myBean" scope="application" class="bean.BreadCrumbBean"
/>
<hbj:breadcrumb id="breadcrumb"
    tooltip="Click to move back to ..."
    onClick="goToPage"
    size="SMALL"
    behavior="DEFAULT"
<!-- model is specified with the id used in useBean (myBean) --%>
<!-- and the property model --%>
    model="myBean.myDefaultListModel"
/>
```


Special Naming Convention - model

The method name `model` invokes a special process. If you define `getModel` and `setModel` methods in your bean you can refer in the JSP to your bean like that:

```
<jsp:useBean id="myBean" scope="application" class="bean.BreadCrumbBean"
/>
    <hbj:breadCrumb id="breadCrumb"
        tooltip="Click to move back to ..."
        onClick="goToPage"
        size="SMALL"
        behavior="DEFAULT"
    <!-- model is specified with the id used in useBean (myBean)      --%>
    <!-- and the property model                                       --%>
        model="myBean.model"
    />
```

Because of an internal convention that becomes effective when using the name `model`, the scope of the `useBean` statement is omitted and the model will be looked up in every scope in the following order:

- page**
- request**
- session**
- application**

In case you want to use different models that you transfer in different contexts (for example, session and application) you have to use another name for the get and set methods.

This special case however is only in effect when you use the `model` tag as shown in the example above (`model="myBean.model"`). If you refer to your model in a scriptlet, the model is taken from the scope declared in the `useBean` statement only.

3.4.2.10 JavaScript API

Purpose

The JavaScript API allows access to HTMLB components in JavaScript programs. Every component which is capable of client side eventing, has to set the `jsObjectNeeded` attribute (inherited from the [Component \[Page 33\]](#) component) to `TRUE` so that the component can be accessed with the JavaScript API. See also the [EventValidationComponent \[Page 33\]](#) description for more details.

Example for InputField

Using the `InputField` component

Uniform Resource Name (URN)

```

Form form = (Form) this.getForm();
InputField inf2 = new InputField("currencyDisplay");
// Enable JavaScript object generation for access in JavaScript
inf2.setJsObjectNeeded(true);
inf2.setClientEvent(EventTrigger.ON_CHANGE,
                    calculateCurrencyToFrom());

inf2.setBCD("100");
inf2.setWidth("250px");
form.addComponent(inf2);

```

Getting the InputField object in JavaScript

```

function calculateCurrencyFromTo() {
    var funcName = htmlb_formid+"_getHtmlbElementId";
    func = window[funcName];
    var inputfield = eval(func("currencyDisplay"));
    // Now we have the inputField, so we can access it, for example
    // set a new value
    if (inputfield)
        inputfield.setValue("100.23");
}

```

Getting the InputField ID with Java

The IdD of a HTMLB component is generate by the HTMLB API. To get the Id of a component use:

```

InputField inf2 = new InputField("currencyDisplay");
String inputfieldID = pageContext.getParamIdForComponent(inf2);

```

The inputfieldID can now be used to access the component:

```
<script> var inputfield = eval(inputfieldID);</script>
```

Components

The components that support client side eventing, like inputField and label, have a API for JavaScript that allow access and modification of the component on the client side.

Label

- **enabled**

A boolean value that indicates if the component is enabled (=true) or disabled (=false).

- **id**

Identification code of the component.

Properties	Methods
id	setEnabled()
isEnabled	setDisabled()

Uniform Resource Name (URN)

Button

- **enabled**

A boolean value that indicates if the component is enabled (=true) or disabled (=false).

- **id**

Identification code of the component.

Properties	Methods
id	setEnabled()
isEnabled	setDisabled()

InputField

- **enabled**

A boolean value that indicates if the component is enabled (=true) or disabled (=false).

- **id**

Identification code of the component.

- **value**

Content of the inputField.

Properties	Methods
id	setEnabled()
isEnabled	setDisabled()
	setValue(value)
	getValue()

CheckBox

- **checked**

A boolean value that indicates if the component is checked (=true) or disabled (=false).

- **enabled**

A boolean value that indicates if the component is enabled (=true) or disabled (=false).

- **id**

Identification code of the component.

Properties	Methods
id	setEnabled()
isEnabled	setDisabled()
	setChecked(boolean)
	getChecked()

Uniform Resource Name (URN)

DropDownListBox & ListBox

- **checked**
A boolean value that indicates if the component is checked (=true) or disabled (=false).
- **enabled**
A boolean value that indicates if the component is enabled (=true) or disabled (=false).
- **id**
Identification code of the component.
- **index**
Index of selected entry of the list box, "top down", starting with 0.
- **option**
An entry in the list box.
- **value**
The key of an entry in the list box.

Properties	Methods	Comment
id	setEnabled	
isEnabled	setDisabled	
	setIndex(index)	Set the selected entry by index number.
	getIndex()	
	addOption(key, visibleText)	
	removeOption(key)	
	setValue(Key)	Sets the selected entry by key.
	getValue()	Get key of the currently selected entry.

RadioButton

To get the radio button object use:

```
function calculateCurrencyFromTo() {
    var funcName = htmlb formid+"_getHtmlbElementId";
    func = window[funcName];
    var rb = eval(func(htmlb_radiobuttonmodifier+"radiobutton"));
    .
    .
}
```

- **enabled**
A boolean value that indicates if the component is enabled (=true) or disabled (=false).
- **id**
Identification code of the component.

Properties	Methods
id	setEnabled()
isEnabled	setDisabled()

Uniform Resource Name (URN)

TableView

The `htmlbevent` object provides methods to get and set following information:

Method	Description
<code>htmlbevent.obj.getClickedColumn()</code>	Returns the column index of the column that has been clicked.
<code>htmlbevent.obj.getClickedRow()</code>	Returns the row index of the row that has been clicked.
<code>htmlbevent.obj.getClickedRowKey()</code>	Returns the row key of the row that has been clicked.
<code>htmlbevent.obj.getSelectedRow()</code>	Returns the index of the selected row (single select mode).
<code>htmlbevent.obj.getSelectedRows()</code>	Returns an array with the indices of all selected rows (multiple select mode).
<code>htmlbevent.obj.getSelectedRowKey()</code>	Returns the key of the selected row (single select mode).
<code>htmlbevent.obj.getSelectedRowKeys()</code>	Returns an array with the keys of all selected rows (multiple select mode).

Example

```
myTable.setOnClientRowSelection("alert('You clicked on row with key  

    '+htmlbevent.obj.getClickedRowKey());  

    alert('Selected rowkeys are:  

    '+htmlbevent.obj.getSelectedRowKeys())");
```

3.4.2.11 Examples

The HTMLB controls can be used in a `DynPage` or `JSPDynPage` in the portal as well as servlets that run on any other servlet machine. This documentation however covers the usage of HTMLB controls in the portal.

3.4.2.11.1 Building a JSPDynPage

This document describes the principles of a `JSP DynPage` and provides a basic example with JSP and describes which methods have to be implemented. The next step is the event handling of a `JSP DynPage` and the data exchange between the `JSP DynPage` and the JSP.

To work with this document you need a basic understanding of Java Server Pages (JSP). Sun provides JSP documentation. The Portal Runtime (PRT) has made modifications to the JSP standard. For details see Java Server Pages (JSP) Support in the Portal Runtime (PRT).

The example has following steps:

[Creating the JSPDynPage \[Page 33\]](#)

[JSPDynPage event handling \[Page 33\]](#)

[Data exchange between JSPDynPage and JSP \[Page 33\]](#)

[Data Exchange Using a Bean \[Page 33\]](#) (used in the example).

Alternative: [Data Exchange Using the Session Object \[Page 33\]](#)

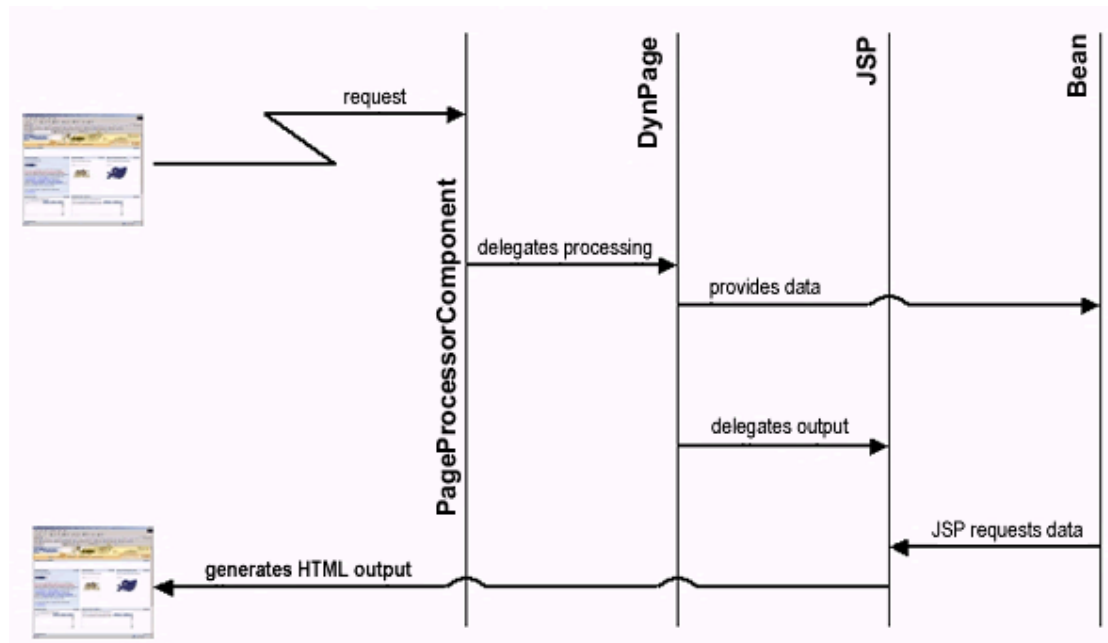
Alternative: [Data Exchange Using the Context Object \[Page 33\]](#)

Alternative: [Data Exchange Using the Request Object \[Page 33\]](#)

Concept of the JSP DynPage

JSP/Servlets offer basic event handling, you have to take care of the event handling yourself (analysing the received form, getting the sender of the event etc.). In addition, the programmer has to take care of the session identifier, which is a unique identifier that makes sure that the datasets are user specific. The JSP DynPage provides enhanced event handling and easy session management. In this example we use the HTML-Business for Java (HTMLB) controls to create the Graphical User Interface (GUI). HTMLB is a Portal service.

Dataflow of a DynPage Component



3.4.2.11.1 Creating the JSPDynPage

First step to create a portal component is to define a class that works as loader class - it inherits from the `PageProcessorComponent`. The created loader class (in the following example named `ExampleOneDyn`) executes the method `getPage()` and returns a unique value of the JSP DynPage we can use (in the following example named `DynPageOne`).

```
package com.mycompany.basicexample;

import com.sap.htmlb.page.DynPage;
import com.sap.portal.htmlb.page.PageProcessorComponent;

public class ExampleOneDyn extends PageProcessorComponent {

    public DynPage getPage() { // Has to be overridden
        // Calls the DynPage and returns its value as DynPageOne
        return new DynPageOne(); }
}
```



The SAP EP Developer Plug-ins provide a JSP Dynpage wizard, that creates all necessary classes, beans and JSP for a JSP Dynpage.

The class `DynPageOne` is extended from the `DynPage` class. Following methods have to be overwritten:

- `doInitialization`

Called when the application is started. The call is made when the page is directly called per URI without parameters and no event occurred.

Usually this method is used to initialize data and to set up models. Be aware of the fact that the `doInitialization` event is also caused when another portal component on the same page sends an event.



With the "Personalize" Dialog you can compose a page by grouping several portal components together. We have created a page called `myPage` with two portal components - A and B. When calling the page `myPage` the `doInitialization` is called from portal component A and B followed by the call of the method `doProcessBeforeOutput`. When an event occurs in the portal component B (for example, by clicking on a button), the `doInitialization` method in portal component A is called again, while in portal component B the method `doProcessAfterInput` followed by the event handling method assigned for the button and finally the `doProcessBeforeOutput` method.

To create solid portal components you must be aware of the fact and check in the `doInitialization` method if the data has already been initialized or the models have been created. Otherwise your portal component is always reset to the initial state if an event in another portal component occurs.

The Enterprise Portal treats every portal component isolated. In this case an event in one portal component does not cause the `doInitialization` event in the other portal component on the same page. The PDK can emulate this behavior by setting the `ISOLATED` flag in the page description XML file to true. The page description XML file is in the content folder of the page and defines the position of the portal component, height and tray type.

Uniform Resource Name (URN)

Example for an entry in the XML file:

```
<component name="AAA.default" title="AAA.default"
height="400" trayType="SAPTrayD3" Position="1"/>
```

Example for an entry in the XML file with isolation flag set so that PDK behaves like the Enterprise Portal:

```
<component name="AAA.default" title="AAA.default"
isolated="true" height="400" trayType="SAPTrayD3"
Position="1"/>
```

If you use the Page Editor in the DevTools section of the PDK you can set the isolated flag interactively.

- **doProcessAfterInput**
Called when the web client sends the form to the web server. Except on doInitialization (see above) the call is performed every time an event occurs.
- **doProcessBeforeOutput**
Called before the form is sent to the web client. The call is performed every time even on doInitialization.

In our example we only use the doProcessBeforeOutput method, the other methods stay "empty".

Uniform Resource Name (URN)

```

package com.mycompany.basicexample;
import com.sap.htmlb.*;
import com.sap.htmlb.enum.*;
import com.sap.htmlb.page.PageException;
import com.sap.portal.htmlb.page.JSPDynPage;

public class DynPageOne extends JSPDynPage {

    /* Constructor */
    public DynPageOne() {
        this.setTitle("DynPageOne");
    }

    /* Used for user initialization. Called when the application is
       * started */
    public void doInitialization() {
    }

    /*
     * Used for handling the input. Generally called each time when an
     * event occurs on the client side.
     */
    public void doProcessAfterInput() throws PageException {
    }

    /* Used for handling the output. This method is always called.
       In our example the JSP makes a textView that displays
       "May the force be with you unknown user". */
    public void doProcessBeforeOutput() throws PageException {
        // set the JSP which builds the GUI
        this.setJspName("OutputText.jsp");
    }
}

```

JSP - OutputText.jsp - that is called by doProcessBeforeOutput

```

<!-- OutputText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
    id="myContext">
    <hbj:page
        title="An Easy Start">
        <hbj:form>
            <hbj:textView
                id="welcome_message"
                text="May the force be with you unknown user"
                design="HEADER1"
            />
        </hbj:form>
    </hbj:page>
</hbj:content>

```

Deployment descriptor: Necessary entries to execute this portal component:

Uniform Resource Name (URN)

```

<application>
  <application-config>
    <property
      name="SharingReference"
      value="htmlb"/>
    </application-config>
  <components>
    <component
      name="default">
      <component-config>
        <property
          name="ClassName"
          value="com.mycompany.basicexample.DynPageOne"/>
        <property
          name="SecurityZone"
          value="com.sap.pct.pdk/low_safety"/>
        </component-config>
        <component-profile>
          <property
            name="tagLib"
            value="/SERVICE/htmlb/taglib/htmlb.tld"/>
          </component-profile>
        </component>
      </components>
    <services/>
  </application>

```

The entry for `ClassName` is case sensitive. The entry `SharingReference` makes sure, that the necessary HTMLB libraries are found.

The strength of the JSP `DynPage` is the event handling. The JSP `DynPage` follows the concept of Java controls (for example, Swing) - Java controls, like the HTMLB controls, can have one or more events. You define the event by assigning a method name to the event. The method is called whenever the event is raised (for example, when a button is clicked). The event handling method is coded in the JSP `DynPage`. The JSP `DynPage` does the event handling and calls the proper event handling method.

Next step in our example is to place a button to our user interface and define an event for it.

3.4.2.11.1.2 JSPDynPage Event Handling

Some HTML-Business for Java controls have an event attribute (for example, see button). The button for example has an 'onClick' attribute that specifies the name of the method which should handle the event. The event will occur when the appropriate user action takes place - in this case clicks on the button. The name of the method specified with the 'onClick' attribute has to be declared in the JSP `DynPage`.

HTML-Business for Java: Statement to specify the method name:

```
Button1.setOnClick("myClick");
```

JSP `DynPage`: Declaration of the method that processes the event:

Uniform Resource Name (URN)

```
public void myClick (Event event) { ..coding.. }
```

or

```
public void onMyClick (Event event) { ..coding.. }
```

Both declarations are valid. The decision to use the on... method declaration could be helpful to make it obvious that this method handles an event.



With the on.. declaration method the first letter of the declared event name must be a capital letter.

If both methods are implemented (myClick and onMyClick), only the method myClick, will be called. The method onMyClick will be ignored.

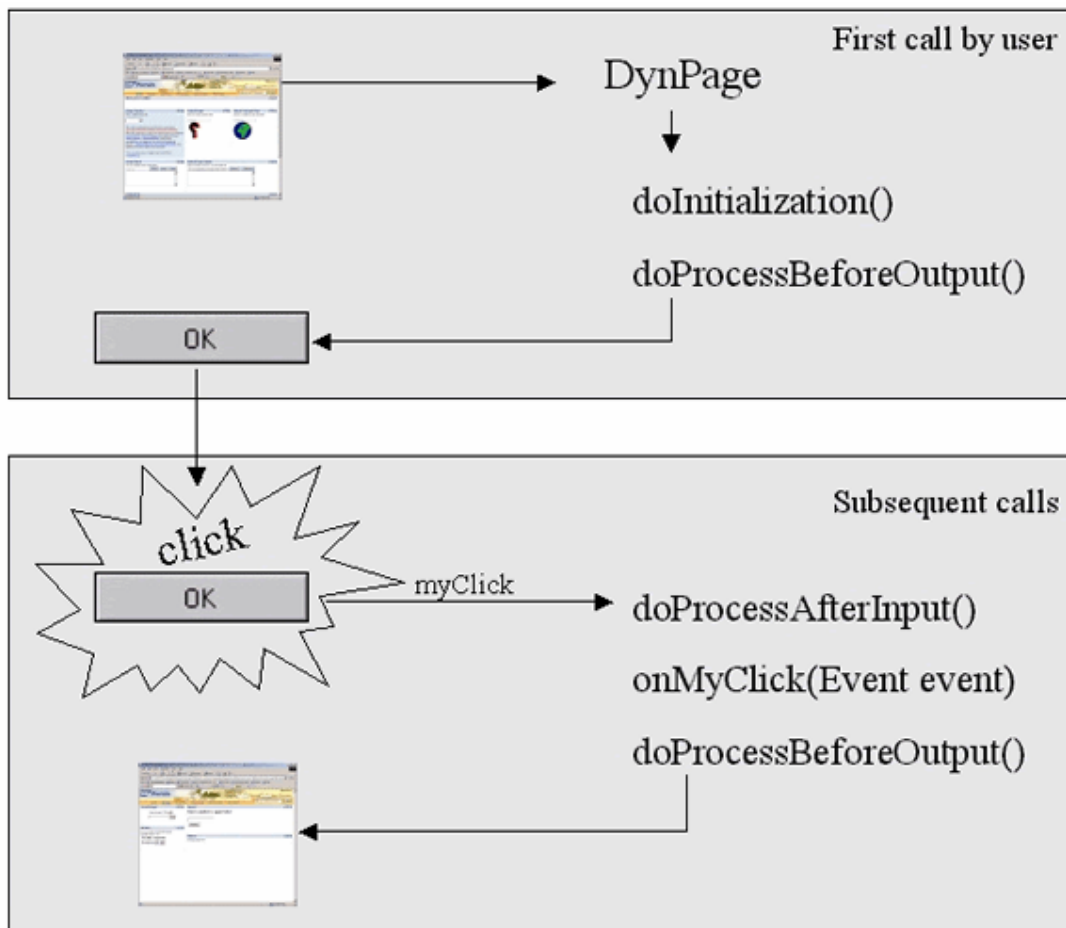


Some HTML-Business for Java controls have a 'setOnClient' attribute. With this attribute you specify a JavaScript fragment that handles the event on the Web client side. The event is NOT transmitted to the Web server.

If an event occurs it is handled as follows (doInitialization is performed when the application starts).

- doProcessAfterInput
Called when the web client sends the form to the web server.
- onMyClick
The event handling method we declared.
- doProcessBeforeOutput
Called before the form is sent to the web client.

Event Processing



3.4.2.11.1.3 Data Exchange between JSPDynPage and JSP

The storing methods we discuss in this section are volatile in the sense that the data is lost when the session is over (or even before that). Generally you have to decide if the data you provide in the JSP DynPage should be shared among other users and how long the data must "live". For storing data permanently you can refer to the Profile documentation.

Storing data can be performed using:

- beans

A bean is defined with set and get methods. The bean can be accessed from the JSP DynPage and the JSP as well as from other users (depending on the scope, see "Usage of Beans").
- session

Data stored in the http session will be kept by the server as long as the user session is alive.
- context

Uniform Resource Name (URN)

The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources.

- request

Data stored in the request are kept for the request.

3.4.2.11.1.3.1 Data Exchange Using a Bean

A bean is used to get and set "dynamic" data. The JSP DynPage usually provides the bean with data and the JSP reads the data. The functionality of the basic example is extended by an input field that allows user input. The user input is stored in a bean and then displayed as text by a JSP program.

Following steps are necessary

- create a bean
- initialize the bean
- introduce the bean to the JSP program `OutputText.jsp`

Declaring a bean in a JSP

The tag [usebean \[Page 33\]](#) declares a bean in a JSP.

- class
Class name of the bean.
- id
Identification name of the bean. The id is used to access the bean in scriptlets.
- scope
Defines the availability of the bean. Details see "[How to use Beans \[Page 33\]](#)".

Attributes	M	Values	Usage – JSP Taglib
class	*	String (cs)	<code>class="com.sap.htmlb.beandemo.myBean"</code>
id	*	String (cs)	<code>id="idOfMyBean"</code>
scope	*	APPLICATION SESSION REQUEST PAGE	<code>scope="APPLICATION"</code>

Bean for the JSPDynPage Example

```
package bean;
/*
 * A simple bean whose only purpose is to store a String.
 * It as a get and set method to store and recall the string.
 */
public class DynPageNameBean {
    public String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

The GUI is extended by an inputField and a button to allow the user to enter a string (for example, user name) and submit the form by clicking the button or pressing Return/Enter on the keyboard.

Following changes to OutputText.jsp are necessary:

- Adding a form to allow the definition of a default button.
- Introduce the bean to the JSP program (<jsp:usebean .../>).
- Changing the textView so that it displays the string retrieved from the bean.
- Adding a label telling the user to enter the user name. The label should start in a new line and one line separated from the textView. That is why you find a

 before the label
- Adding the inputField "user_name_input" - the JSP DynPage retrieves the data in the input field using getComponentByName.
- Adding a send button which we also define as the default button. This enables the user to send his input back to the server by either clicking on the button or by simply pressing Enter/Return on the keyboard when he finished the input.

```
<!-- OutputText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
    id="myContext">
    <hbj:page
        title="An Easy Start">
        <hbj:form
            id="myFormId">
            <!-- Declaration of the bean. -->
            <jsp:useBean
                id="UserNameBean"
                scope="application"
                class="bean.DynPageNameBean"
            />
            <hbj:textView
                id="welcome_message"
                design="HEADER1">
            <%
                welcome_message.setText
                ("May the force be with you "+UserNameBean.getName());
```

Uniform Resource Name (URN)

```
%>

</hbj:textView>
<br>
<br>
<hbj:label
    id="label_input"
    text="Your name please"
    design="LABEL"
    required="TRUE"
    labelFor="user_name_input"
/>

<!-- inputfield to allow userInput - the inputfield has the id --%>
<!-- "user name input" which is used in the JSP DynPage to --%>
<!-- access the input field and retrieve the input of the user --%>
<hbj:inputField
    id="user_name_input"
    type="STRING"
    design="STANDARD"
    width="250"
    maxlength="30"
/>
<hbj:button
    id="Send_Button"
    text="Send"
    tooltip="Sends my name"
    onClick="onSendButtonClicked"
    width="100"
    design="EMPHASIZED">
<%
    myFormId.setDefaultButton(Send_Button);
%>

</hbj:button>
</hbj:form>
</hbj:page>
</hbj:content>
```

Result

Hint: The default string "unknown user" will be set in our JSP DynPage in the following section.

May the force be with you unknown user

Your name please*

Changes in the JSPDynPage

The JSP DynPage has to be adjusted as well. Following steps are necessary:

- Introducing the bean to the JSP DynPage (Import statement).
- In the doInitialization() method we set a default user name to "unknown user".

Uniform Resource Name (URN)

- Creating the event method onSendButtonClicked() for the button click in which the status (variable state) is set to WELCOME_STATE so that the doProcessBeforeOutput selects another JSP file.
- In doProcessAfterInput() method (which is called whenever an event occurs) we request the inputField "user_name_input" from the JSP by using getComponentByName to have access to the user input in the JSP DynPage. If the inputField "user_name_input" is not empty the string is stored in the bean.

```

package com.mycompany.basicexample;
/** introduce the bean */
import bean.DynPageNameBean;
import com.sap.htmlb.*;
import com.sap.htmlb.enum.*;
import com.sap.htmlb.event.Event;
import com.sap.htmlb.page.DynPage;
import com.sap.htmlb.page.PageException;
import com.sap.portal.htmlb.page.JSPDynPage;
import com.sap.portal.htmlb.page.PageProcessorComponent;
import com.sap.portal.prt.component.IPortalComponentContext;
import com.sap.portal.prt.component.IPortalComponentProfile;
import com.sap.portal.prt.component.IPortalComponentRequest;

public class DynPageOne extends JSPDynPage {
    private final static int INITIAL_STATE = 0;
    private final static int WELCOME_STATE = 1;
    private int state = INITIAL_STATE;
    private String name;

    /**
     * Constructor
     */
    public DynPageOne() {
        this.setTitle("Become a Jedi");
    }

    /**
     * Used for user initialization. called when the application is
     * started
     */
    public void doInitialization() {
        // create the bean and set a default text value "unknown user"
        IPortalComponentRequest request =
            (IPortalComponentRequest) this.getRequest();
        IPortalComponentContext myContext = request.getComponentContext();
        IPortalComponentProfile myProfile = myContext.getProfile();
        // new bean object
        UserNameContainer = new DynPageNameBean();
        // set default name
        UserNameContainer.setName("unknown user");
        // store bean in profile for the JSP
        myProfile.putValue("UserNameBean", UserNameContainer);
        // Set the state so that we can decide what action to do next
        state = INITIAL_STATE;
    }

    /**
     * Used for handling the input. Generally called on each event
     * we use this method to get the user name and store it in the bean
     */

```


Uniform Resource Name (URN)

```

public void doProcessAfterInput() throws PageException {
    // get the input field from the JSP
    InputField myInputField =
        (InputField) getComponentByName("user_name_input");
    if (myInputField != null) {
        this.name = myInputField.getValueAsDataType().toString();
    }
    IPortalComponentRequest request =
        (IPortalComponentRequest) this.getRequest();
    IPortalComponentContext myContext = request.getComponentContext();
    IPortalComponentProfile myProfile = myContext.getProfile();
    DynPageNameBean myNameContainer =
        (DynPageNameBean) myProfile.getValue("MyNameBean");
    myNameContainer.setName(name);
}
/**
 * Used for handling the output. This method is always called.
 * In our example before the JSP made a textView with
 * "May the force be with you unknown user".
 * We now extend this method that according to the state it either -
 * that is when state = INITIAL_STATE - asks for the user name
 * and calls the user "unknown user" or after init - that is when
 * state = WELCOME_STATE - displays a success message with
 * the username.
 */
public void doProcessBeforeOutput() throws PageException {
    switch (state) {
        case WELCOME_STATE :
            this.setJspName("OutputSuccessText.jsp");
    }
    break;
    default :
        this.setJspName("OutputText.jsp");
        break;
}
/**
 * this method handles the event of the button. The event is fired
 * either when the user clicks on the button or presses the
 * Return/Enter key when he is in the inputField (since we defined
 * the button as default button). In this method we set the state to
 * WELCOME_STATE so that on the following doProcessBeforeOutput
 * (which is called immediately after this method)
 * a success message is displayed
 */
public void onSendButtonClicked(Event event) throws PageException {
    state = WELCOME_STATE;
}
}

```

The only thing missing now is OutputSuccessText.jsp which should send the personalized message to the user. The usage of the bean has been already shown in OutputText.jsp so OutputSuccessText.jsp is very small and creates a textView with the username in it.

Uniform Resource Name (URN)

```

<!-- OutputSuccessText.jsp -->
<%@ taglib uri= "tagLib" prefix="hbj" %>
<hbj:content
  id="myContext">
  <hbj:page
    title="Successful processing">
    <jsp:useBean
      id="UserNameBean"
      scope="application"
      class="bean.DynPageNameBean"
    />
    <hbj:textView
      id="success_message"
      design="HEADER1">
      <%
        success_message.setText
          ("The force is with you" + UserNameBean.getName());
      %>
    </hbj:textView>
  </hbj:page>
</hbj:content>

```

Result of the JSP

(assuming that the user responded with "SAP")

The force is with you SAP**3.4.2.11.1.3.2 Data Exchange Using the Session Object**

Data stored in the http session will be kept by the server as long as the user session is alive.

JSPDynPage

Getting the request object:

```

IPortalComponentRequest request =
    (IPortalComponentRequest) this.getRequest();

```

To store a value in the session we have to use the command line:

```

request.getComponentSession().putValue("myText",
                                         "Text in the session
context");

```

To get the stored value we have to use the command line:

```

request.getComponentSession().getValue("myText");

```

JSP

To store a value in the session we have to use the command line:

```
<%
    componentRequest.getComponentSession().putValue("myText",
                                                    "That text is from the
JSP");
%>
```

To get the stored value we have to use the command line:

```
<%

    componentRequest.getComponentSession().getValue("myText").toString
(); %>
```

3.4.2.11.1.3.3 Data Exchange Using the Context Object

The lifetime of data stored in the context cannot be guaranteed. The context can be released any time when the web server needs resources. We strongly recommend to refresh the stored data (for example, in the JSP DynPage method `doProcessAfterInput`, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request =
    (IPortalComponentRequest) this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getComponentContext().putValue("myText",
                                        "A short note in the
context");
```

To get the stored value we have to use the command line:

```
request.getComponentContext().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%
    componentRequest.getComponentContext().putValue("myText",
                                                    "From the JSP");
%>
```

Uniform Resource Name (URN)

```
%>
```

To get the stored value we have to use the command line:

```
<%  
  
componentRequest.getComponentContext().getValue("myText").toString  
();  
  
%>
```

3.4.2.11.1.3.4 Data Exchange Using the Request Object

The lifetime of data stored in the request is limited to the request only. You have to refresh the stored data (for example, in the JSP DynPage method `doProcessAfterInput`, which is called every time an event occurs on the web client) to avoid an exception. The exception will occur when you try to access a value that is already gone (or has never been set).

JSP DynPage

Getting the request object:

```
IPortalComponentRequest request =  
    (IPortalComponentRequest) this.getRequest();
```

To store a value in the session we have to use the command line:

```
request.getNode().putValue("myText", "A short note in the  
request");
```

To get the stored value we have to use the command line:

```
request.getNode().getValue("myText");
```

JSP

To store a value in the session we have to use the command line:

```
<%  
    componentRequest.getNode().putValue("myText", "That is from the  
JSP");  
%>
```

To get the stored value we have to use the command line:

```
<%  
    componentRequest.getNode().getValue("myText").toString();  
%>
```

3.4.2.12 Mobile Enhancements

Purpose

Mobile enhancements for SAP NetWeaver support the development of user interfaces for mobile devices such as pocket PCs or WAP-capable mobile telephones, thereby enhancing the SAP Web infrastructure. They are based on an online technology in which the mobile device is directly linked to a Web server. Examples of these technologies include WAP, wireless LANs, Bluetooth, or GPRS. The mobile enhancements enable you to create and modify mobile Web applications simply and cost-efficiently.

3.4.2.12.1 Mobile HTML-Business for Java

Purpose

The following documentation provides your development team with an introduction to the technical extensions to HTMLBusiness for Java for mobile devices, such as pocket PCs and WAP-enabled devices. These mobile extensions allow developers to create Web applications with easy to use Web controls. Moreover, using the device recognition mechanism described by the [ClientInfo Package \[Page 33\]](#), they also provide detailed information about the mobile device making the request and its browser capabilities. These browser capabilities are taken into account when Web controls are displayed on the mobile devices.

For more information, refer to:

- [Mobile Extensions to the Java Servlet Containers \[Page 33\]](#).

3.4.2.12.1.1 Mobile HTML-Business for Java – An Overview

The mobile extensions to HTMLBusiness for Java allow you to develop Web applications for mobile devices in a **device-independent** way. The programming effort associated with developing Web applications for different mobile device types can thus be simplified and reduced.

These extensions not only enable development teams to create Servlets and Java Server Pages (JSPs) **independent of devices and platforms**. They also remove the need for developers to learn the markup language WML, which would otherwise be necessary to develop applications for WAP-enabled devices. To keep you abreast of the latest development techniques, SAP will include new markup languages, such as XHTML, immediately. Another advantage of Mobile HTMLBusiness for Java is that SAP has already taken the differences between micro-browsers on different mobile devices into account. It is important to remember that this includes differences in the way Web pages are displayed in the same browser on different devices.

Introductory notes

Mobile HTML-Business for Java is implemented in the package `com.sap.mobile.htmlb`. To run Mobile HTMLBusiness for Java, you must install (that is, deploy) the two packages

Uniform Resource Name (URN)

`com.sap.mobile.clientinfo` and `com.sapportals.htmlb` in the Java Servlet Engine environment.

The ClientInfo package contains the device recognition data of the different mobile devices and implements the device recognition process. For more information on Web application client information, refer to [Device Recognition \[Page 33\]](#) and [ClientInfo Interface \[Page 33\]](#).

The Client package is delivered in the archive files `clientinfo.jar`, `clientinfo.war`, `clientinfo.ear`, and `clientinfo.sda`. The Mobile HTMLBusiness for Java package is stored in the archive files `mobilehtmlb.jar`, `mobilehtmlb.war`, `mobilehtmlb.ear`, and in the SAP-specific format `mobilehtmlb.sda`.



The Software Delivery Archive (SDA) is the delivery format for SAP applications in languages other than ABAP. It is a ZIP-compatible archive format used as a container for other archives. The SDA contains the manifest information for the archives it contains (such as `.jar`, `.war`, and `.ear` files) along with a SAP manifesto containing additional information needed for software logistics.

Integration

To be able to use the functions of Mobile HTMLBusiness for Java you need the two packages `com.sapportals.htmlb` and `com.sap.mobile.clientinfo`.

Features

HTMLBusiness for Java lets you create Web pages with the emphasis on design and provides application programmers with a range of basic Web controls (components) in the form of a class and tag library – similar to the Swing classes offered by Java.

Mobile HTMLBusiness for Java extends the functions of HTMLBusiness for Java for mobile devices.

This mobile extension thus enables you to use Web controls to develop Web applications for mobile devices like pocket PCs and WAP-enabled mobile telephones.

The term Web controls refers to user interface elements that are used in Web applications to allow users to execute actions. Pushbuttons, checkboxes, and radio buttons are just a few examples of Web controls. They can be inserted in servlets and JSP pages using specially-provided tags.

As a subclass of HTMLBusiness for Java, Mobile HTMLBusiness for Java uses all its functions – not just the Web controls themselves, but also:

- The runtime environment
- Event handling (creating events, when the user performs an action, such as moving the mouse)
- The data binding concept (access to data)

The development team no longer needs to go into the different behaviors of mobile devices and their Web browsers. (The differences between Web browsers used by mobile devices are even greater than those between browsers used on personal computers). In some cases, difference are so great that you need to write several different Web applications to optimize display on different devices. (See [Differences Between Mobile Devices \[Page 33\]](#)).

3.4.2.12.1.2 Model View Controller Architecture – The Underlying Concept

The HTMLBusiness Library is based on the Model View Controller (MVC) architecture, which also provides the basis for the Swing components in Java.

In this approach, the programming model for user interface elements is divided into three areas:

- **Model** = the component (Web control)
- **View** = the graphical display (renderer)
- **Controller** = the object that calls the correct renderer class

In HTMLBusiness for Java, there is a model class for each component (Web control, container) that can be accessed through its interface. In addition, there is at least one view class for each component, which translates the components into actual HTML. The central controller class, `PageContext`, and the interface `IPageContext` control the associated view classes. When a mobile device sends an HTTP request, this class ensures that the renderer class used is the one for this device.

Renderer classes are pseudo-static in character. That is, like static classes, they do not create instances, whereas like dynamic classes they are not instantiated until runtime and do not reserve space in memory till then. They create an instance for each browser and each Web control.

Within this MVC architecture, Mobile HTMLBusiness for Java supports all components (Web controls) relevant for mobile devices – such as buttons or tableViews – and provides special view classes (renderer classes) for mobile devices such as pocket PCs and WAP-enabled devices. In this way Mobile HTMLBusiness for Java provides a range of Web controls for mobile devices allowing you to develop Web applications that can be run and displayed on different devices.

3.4.2.12.1.3 Special Features of Web Controls for Mobile Applications

Overview

Our main aim is to enable you to develop Web applications without your having to take the properties of different devices and browser flavors into account. This means that the source code you write for a Web application for a conventional personal computer can also run without additional problems on mobile devices.

Use

Using Web controls, you can program Web applications quickly and easily. You need not worry about the different ways in which this application will be displayed. Each Web Control possesses different attributes that define the appearance of that control on the screen.

(For examples, see the *User manual for HTML-Business for Java* documentation).

There are mobile devices that cannot evaluate some of these attributes and thus cannot support their display on screen. When you use Mobile HTMLBusiness for Java, these attributes are ignored. The **button** Web control possesses the attributes `design`, `disabled`, `id`, `text`, `width`, and `tooltip`. For pocket PCs for example, the value of the *design* button attribute is limited to the value *Standard*, since you cannot display buttons highlighted on this device

Uniform Resource Name (URN)

(among other limitations). The *Emphasized* and *Small* values are ignored and always set to the predefined value *Standard*.

When you use Web controls on mobile devices, the following principles generally apply:

- The following Web controls are supported for Web applications on mobile devices: Button, breadCrumb, checkbox, checkboxGroup, dateNavigator, dropdownListBox, gridLayout, group, image, inputField, itemList, label, link, listBox, radioButtonGroup, tableView, tabStrip, textEdit, textView, and tray.
- The *tooltip* and *disabled* attributes are ignored for all available Web controls.
- All server-side events – such as onClick – are supported.



Client-side events such as onClientClick are not supported for mobile devices.



If you want to use the **image** Web control for WAP-enabled devices, you must also save the graphic in the WBMP format, supported by WAP browsers, with the same filename and in the same directory – for example src="/sap/Walldorf.jpg" and src="/sap/Walldorf.wbmp"



If an attribute is not supported, the application takes its default value. (For the default value of each control attribute, refer to the documentation on HTMLBusiness for Java).

The following table shows further constraints for specific Web controls when used with mobile applications. It also shows which events, attributes, and attribute values are ignored when specific Web controls are used on mobile devices, along with the effects of the attribute when the control is displayed on screen.

(**Unsupported** values are indicated with a cross).

Component	Attributes/events	Values	HTML	WAP
button	design	STANDARD		
		SMALL	X	X *
		EMPHASIZED	X	X *
	disabled	TRUE	X	X
		FALSE		
form	target	_blank	X	X
		_self		
		_parent	X	X
		_top	X	X
gridlayout	style	STRING	X	X
image	src			See note above
inputfield	design	STANDARD		

Uniform Resource Name (URN)

	invalid	SMALL FALSE	X	X *
	visible	TRUE FALSE	X X	X X
label	design	LABEL LABELSMALL		X *
link	target	_blank _self _parent _top	X X X	X X X
tabStrip	bodyheight	Unit	X	X
textView	design	EMPHASIZED HEADER1 HEADER2 HEADER3 LABEL LABELSMALL LEGEND REFERENCE STANDARD		X *
tray	onEdit onRemove	String String	X X	X X

Note:

Support for this attribute value for WAP-enabled devices depends on the device itself. Some devices support the WML tabs <small>, , and while others do not. If a WAP-enabled device supports the specified WML tag, the associated values *Small*, *Emphasized*, *LabelSmall*, and so on are also supported.

3.4.2.12.1.4 Using JSP Tags and Components Similar to Beans

Overview

Mobile HTMLBusiness for Java offers a range of reusable interface elements (Web controls) that you can use when programming user interfaces. It makes sense to use these Web controls if you want to develop Java applications for different browser flavors (including micro-browsers on mobile devices).

By using Web controls you can substantially reduce the amount of code you need to write (to mention just one benefit of these objects). This does not only improve the performance of mobile devices. In some cases, it makes it possible to run them at all. This is because the amount of memory available to applications is of particular importance in this area. For

Uniform Resource Name (URN)

example, many WAP-enabled devices, like the Nokia 7110 can only handle Web pages that are a maximum of 1.36 Mb in size.

Example

The Mobile HTMLBusiness for Java extension enables users to run Java applications developed for the PC on mobile devices. When developing these applications, you can use the normal tag library for developing Java Server Pages (JSPs) and the Class Library for developing servlets.

The following table shows you how to use the **textView** Web control in different Web components (servlets, JSPs):

Web component	Java code
Servlet	<pre>TextView t = new TextView (); t.setText ("Hello World!");</pre>
JSP	<pre><hbj: textView id="Answer_Required_ZIP" required="TRUE" text="ZIP Code" design="emphasized"> </hbj:textView></pre>

3.4.2.12.1.4.1 Example: Displaying the tableView Web Control

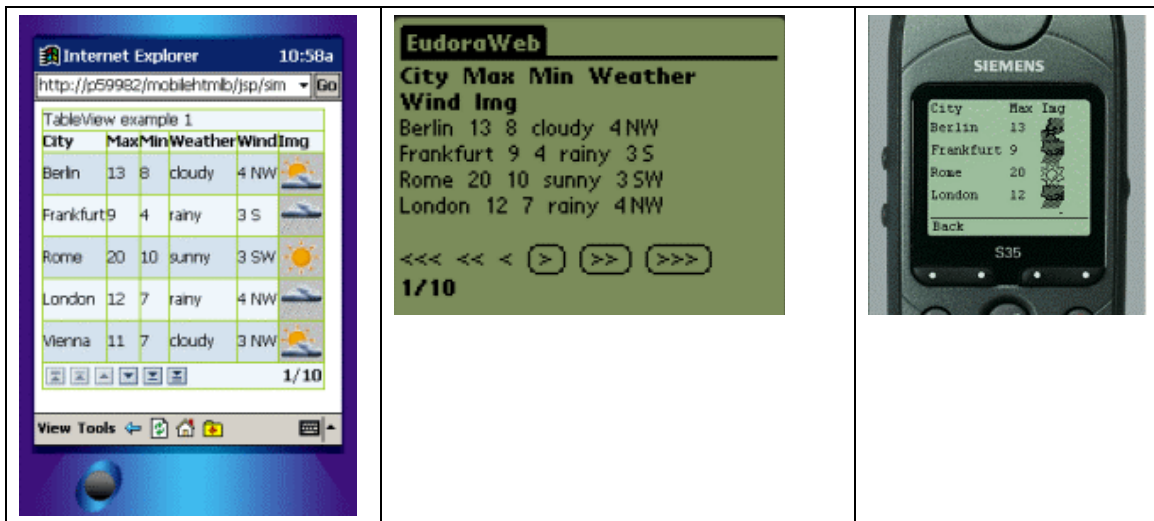
The following example of the **tableView** Web control shows the different possibilities available for displaying information on different mobile devices. This example also makes it clear that the Web controls for each device are displayed correctly, without your having to develop special source code.

The Web controls are displayed for a specific device in such a way that they support the properties and characteristics of each device.

For example, the same JSP source code contained in the **tableView** Web control, is displayed as follows on the following mobile devices (and emulators):

(1) Pocket-PC emulator	(2) Palm emulation with Eudora Browser	(3) WAP emulator
------------------------	--	------------------

Uniform Resource Name (URN)

**JSP example: Source code**

```
<%@ page import="com.sapportals.htmlb.*,com.sap.mobile.htmlb.rendering.*,
com.sapportals.htmlb.table.TableView,com.sapportals.htmlb.event.*" %>

<%@ taglib uri="htmlb.tld" prefix="hbj" %>
<hbj:content id="myContext" >
<hbj:page title="TableView tag">
<hbj:form id="myFormId" method="post" action="">
<jsp:useBean id="myTableViewBean" scope="page"
class="com.sap.mobile.htmlb.test.MobileJspBean" />
<%
if( myContext instanceof MobilePageContext )
myTableViewBean.model.setPageContext((IMobilePageContext)myContext);
%>
<hbj:tableView id="myTableView1"
model="myTableViewBean.model"
design="ALTERNATING"
headerVisible="true"
footerVisible="true"
fillUpEmptyRows="true"
navigationMode="BYLINE"
headerText="TableView example 1"
onNavigate="myOnNavigate"
visibleFirstRow="1"
visibleRowCount="5"
rowCount="20"
>

<% // EventHandler for tableView
Event myEvent = myContext.getCurrentEvent();
if (myEvent != null) {
if (myEvent instanceof TableNavigationEvent) {
TableView table = (TableView)myContext.getComponentForId("myTableView1");
TableNavigationEvent event = (TableNavigationEvent)myEvent;
myTableView1.setVisibleFirstRow(event.getFirstVisibleRowAfter());
}
}
```

Uniform Resource Name (URN)

```
%>
</hbj:tableView>
</hbj:form>
</hbj:page>
</hbj:content>
```

Weather data binding: Source code

```
package com.sap.mobile.htmlb.test;

import com.sap.mw.jco.JCO;

import com.sapportals.htmlb.table.TableColumn;
import com.sapportals.htmlb.enum.TableColumnType;
import com.sapportals.htmlb.table.TableViewModel;
import com.sap.mobile.htmlb.table.MobileTableViewModel;

public class MobileJspBean {
    static String[][] strWeather =
    {
        { "Berlin", "13", "8", "cloudy", "4 NW", "cloudy.jpg" },
        { "Frankfurt", "9", "4", "rainy", "3 S", "rainy.jpg" },
        { "Rome", "20", "10", "sunny", "3 SW", "sunny.jpg" },
        { "London", "12", "7", "rainy", "4 NW", "rainy.jpg" },
        { "Vienna", "11", "7", "cloudy", "3 NW", "cloudy.jpg" },
        { "Lisbon", "23", "12", "sunny", "3 NE", "sunny.jpg" },
        { "New York", "8", "2", "cloudy", "1 N", "cloudy.jpg" },
        { "Los Angeles", "24", "14", "sunny", "2 SE", "sunny.jpg" },
        { "Kapstadt", "20", "17", "cloudy", "4 S", "cloudy.jpg" },
        { "Peking", "17", "2", "sunny", "3 S", "sunny.jpg" },
    };

    static Object[] colnames = {
        "City", "Max", "Min", "Weather", "Wind", "Img" // header
    };

    static int[] colprios = {
        1, 1, 2, 2, 2, 1 // header
    };

    // Properties:
    public MobileTableViewModel model;

    public TableViewModel getModel() {
        return this.model;
    }

    public void setModel(MobileTableViewModel model) {
        this.model = model;
    }

    // Default constructor:
    public MobileJspBean() {

        String mimePath = "/mobilehtmlb/mimes/";
        int i=0;
        while( i<10 ) {
            if( strWeather[i][5].indexOf('/') == -1 )
                strWeather[i][5] = mimePath + strWeather[i][5];
            i++;
        }
    }
}
```

```
model = new MobileTableViewModel(strWeather, colnames, colprios);
TableColumn col6 = model.getColumnAt(6);
col6.setType(TableColumnType.IMAGE);
}
}
```



We used emulators to consider the display of JSPs. Emulators are programs that simulate the behavior of real devices. They are particularly well-suited to testing applications, above all if you do not have the device available.

As you can see, the example of the weather information table looks different on different browsers. However, it has been displayed on the screen according to the supported browser characteristics.

- (1) A pocket PC with a color screen can display pictures in .jpg or .gif format along with gridlines, columns and rows in a tabular form, with appropriately-aligned cell contents.
- (2) The Eudora Browser used on Palm devices can display table rows, but cannot put data in vertically-aligned columns.
- (3) Conversely, the WAP-enabled mobile telephone from Siemens can display tables in the expected way. It also supports pictures in wireless bitmap (WBMP) format. However, this mobile telephone can only realistically display three columns, since it has a relatively small screen. You can specify the number of columns that a mobile device should display yourself using priorities (see above in the highlighted source code for weather data binding). To do this, the **MobileTableViewModel** class is extended.



However, other WAP-enabled mobile telephones display tables differently (see [isTableSupported Method \[Page 33\]](#) of the ClientInfo Interface).

3.4.2.12.2 Mobile Extensions to the Java Servlet Container

Purpose

The mobile extensions developed for a Java-enabled Web server and implemented in the package `com.sap.mobile.clientinfo` allow you to develop device-specific Web applications for mobile devices in the Java programming language.

By using these extensions, the attributes and capabilities of the mobile device making the request can be specified, and then taken into account when the Web application is displayed. Moreover, the extensions support the use of Web controls when developing Web applications that can run on mobile devices, allowing you to program Web applications that are browser and device-independent.

Features

If a Java application is to be developed, using a Java Server Page (JSP) or servlet, for example, the development team can also access the mobile extensions of the Java Servlet

Uniform Resource Name (URN)

Engine. These mobile extensions make available the special requirements and properties of mobile devices, such as:

- Size of the display field
- Input methods
- Markup languages
- Browser variants
- JavaScript support

The mobile extensions refer to the use of mobile devices like WAP-enabled mobile telephones and Personal Digital Assistants (PDAs) and consist of the `ClientInfo` interface, implemented using the SAP device recognition process.

The `ClientInfo` interface describes methods that provide the special properties of mobile devices, and returns information on browser characteristics on these devices.

You can use this interface to create device-specific Web applications for mobile devices in the Java programming language. You can thus take the browser and device-specific properties into account when constructing the user interface.

Example

The `isTitleSupported` method of the `ClientInfo` interface specifies whether or not a label is to appear on the top of the screen using the “title” attribute of the `<card>` WML tag. On some devices, entering a title in the `<card>` WML tag displays a header in the Web application title bar. However, many types of browser do not support this function. If they do not, no header appears in the device’s user interface. To make the title appear on such devices, it must be included in the body text of the document.

(See also [isTitleSupported Method \[Page 33\]](#)).

3.4.2.12.2.1 Differences Between Mobile Devices

The differences between devices can be split into the following categories:

- Size of the display field
- Input method
- Markup language
- Browser variant

Even when you use the standardized language WML, the differences are so great that you need to create different formatting for the page for different devices, to optimize display. The following sections list the differences in more detail:

Categories

- Size of the display field
 - PDA or Smartphone
 - PDA or Smartphone with VGA/4 format – such as pocket PC, Palm (similar to VGA/4)
 - Telephones with a small screen

Uniform Resource Name (URN)

- Input method
 - Keypad
 - On-screen input using keyboard or handwriting recognition – for example, pocket PC, Palm
 - Telephone keypad
- Markup language
 - HTML 3.2 – for example, pocket PC
 - Restricted version of HTML – for example, Palm, i-mode, Mobile Explorer
 - WML (>=Version 1.1)
- Browser variant
 - WAP 1.0
 - WAP 1.1 – Nokia
 - WAP 1.1 – Openwave Browser
 - WAP 1.1 – Ericsson
 - WAP 1,2
 - WAP browser on PDAs (EZOS, Materna, WAPMan, and so on)



The differences in the display of an Internet service depend not only on the browser variant. The same browser on different devices displays the Internet service differently.

Other differences

- Color display capability (color, grayscale, black and white)
- Graphic formats

See also:

[Effects of the Differences Between WAP Browsers \[Page 33\]](#)

3.4.2.12.2.2 Effects of the Differences Between WAP Browsers

The following examples show why different WAP browsers require different WML pages. The following list is sorted in descending order of priority.

1. Ability to display a page

The same page is displayed correctly on one browser but cannot be displayed at all on another.



In many cases, pages on one manufacturer's browser are displayed completely, whereas they are cut off on a browser from a different manufacturer. For this reason, you must test your pages on browsers from several different manufacturers.

2. Usability of functions

Uniform Resource Name (URN)

Certain functions cannot be used on certain browsers.



WTAI-Links (Wireless Telephony Application Interface) work on Openwave Browsers. On some devices, the links are displayed but do not work. For this reason, you should not send WTAI links to the Nokia 7110 mobile telephone, for example. Conversely, you should use them on Openwave Browsers.

3. Display quality (1)

The appearance of the page is unacceptable.



The markup language WML contains the table tag `<table>`. The Nokia 7110 interprets this, but does not do justice to its tabular character. On an EZOS Browser, the table is formatted correctly.

4. Display quality (2)

Many of the differences between browsers cause the display of the same page to differ on different browsers, but in a way that is still acceptable.



For example, the character formatting tags ``, `<u>`, `<small>`, and so on are ignored by the Nokia 7110 mobile telephone. However, this does not affect the user's ability to use the functions. You should use these tags in pages displayed on an Openwave Browser, since they make the page more readable.

The above are only a few of the differences between different browsers. For this reason, it is unreasonable to expect application developers to know about all the differences that exist and to take account of them in their applications.

3.4.2.12.2.3 Device Recognition

Purpose

The device recognition mechanism is part of the device-specific development of Web applications. It ascertains the device type from the HTTP query.

Prerequisites

You are using a Java-enabled Web server and have installed a Java Servlet Engine on your Web server. In addition, you have adapted the configuration for an enterprise-specific deployment.

Process Flow

Identifying a mobile device

The process of device recognition is based on the HTTP request headers "Accept" and "userAgent", which are sent to the server in the HTTP request by the requesting device. They enable the application to access the special capabilities of the requesting device through the implementation of the `ClientInfo` interface.

Uniform Resource Name (URN)

The device recognition mechanism compares the character sequence in the “userAgent” HTTP request header – for example, Nokia7110/1.0 (04.84), (information needed to identify the device) – with the values of the device properties in the devices.xml configuration file.

If it finds this character sequence in the file, the associated values can be accessed using the `ClientInfo` interface.

When evaluating the HTTP request header, the exactness with which the device is identified is taken into account. To do this, a specific priority value is assigned to each device type in the XML <priority/> tag in the configuration table.

If the priority value = 1, the character sequence matches exactly and a specific device type has been ascertained. If it cannot find a match when comparing the sequence, the device recognition mechanism tries to assign the device to a superordinate device type of priority 2 or 3.

For example, if the HTTP request “userAgent” contains a sequence beginning with “Nokia”, the device type is recognized as a Nokia device and is assigned the values of a *NokiaGeneric* device type. The mechanism can invoke this generic device type, since developers can generally assume that a manufacturer will use the same browser for all devices.

If the device recognition mechanism cannot find any match in the configuration table, it can assign one of two types to the device, based on the value in the HTTP request header “Accept”: Either the *wmlGeneric* type for WAP-enabled devices or the *htmlGeneric* type for HTML-enabled mobile devices. In this way, the device recognition mechanism can identify a mobile device even if there is no entry for it in the devices.xml configuration file.

How exactly the `ClientInfo` interface provides the specific values of the properties of a device is dealt with in [The ClientInfo Interface in the Java Servlet Engine \[Page 33\]](#).

Result

The device recognition process ascertains the type of the device and provides a set of properties specific to the mobile device making the request.

3.4.2.12.2.4 The ClientInfo Interface in the Java Servlet Engine

Use

The device recognition mechanism in the Java Servlet Engine environment ascertains the device type from the HTTP request and enables access to the special properties and capabilities of the mobile device.

Implementing the device recognition process

The [ClientInfo Interface \[Page 33\]](#) offers a range of methods that investigate the different display capabilities of different browser types. The values of the device properties provided by SAP for a variety of available devices are stored in XML files on the Java-enabled Web server.

The device properties of a specific device type are made available in a XML file with the extension **.cap**. This XML file contains all the properties of the mobile device that provide information on its capabilities, along with properties for which SAP proposes default values for a specific device.

Uniform Resource Name (URN)

The .cap XML files for all the device types entered by SAP are stored on the Java-enabled Web server. They allow system administrators to perform the following tasks within the device recognition process:

- [Modify Device Properties \[Page 33\]](#)
- [Integrate New Mobile Devices \[Page 33\]](#)



The XML files are stored in <Root_Directory>/mobile.

When the Java Servlet Container is launched, the *ClientInfoInitializer* servlet is executed. This servlet loads the data from the XML file and stores the Java tables for the device properties in the Web server's memory. It thus provides access to the special values of the properties of the requesting mobile device.

Using the device type information sent in the HTTP request, the system selects the entry appropriate to the requesting device type from the table. This allows the values for this device to be ascertained and made available at runtime when a Java application is launched.

In this way, you can use the methods of the *ClientInfo* interface when you create Java applications, and thus take the characteristics of mobile devices into account when developing Web applications.

You can use the following source code fragments to access the *ClientInfo* interface and device recognition mechanism:

```
ClientInfoFactory factory = ClientInfoFactory.newInstance();
ClientInfo clientInfo = factory.newClientInfo();
clientInfo.load(request);
if(clientInfo.isTitleSupported())
    // do sth if title is supported
    ...
else
    // do sth if title is not supported
    ...
```

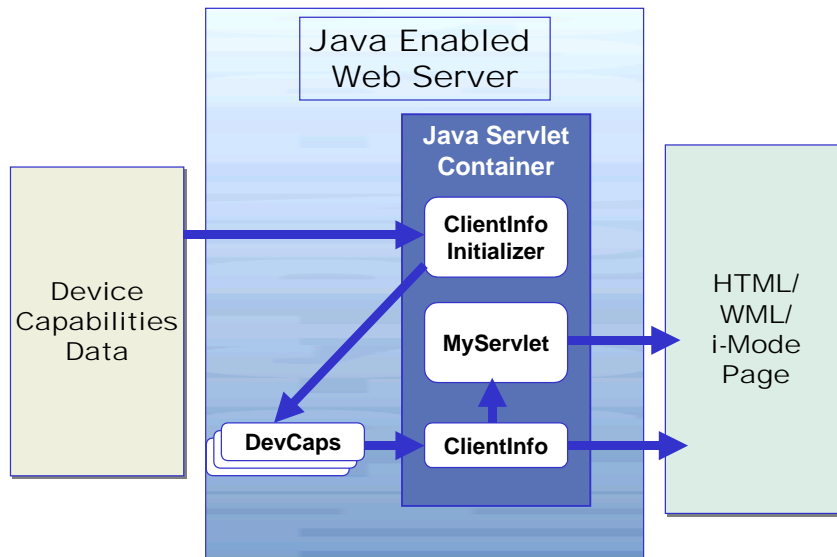
In these fragments, an implementation of the *ClientInfo* interface is instantiated and returned as the *ClientInfo* interface. The *newClientInfo* method of the *ClientInfoFactory* class returns the class implemented using the *ClientInfo* interface. This is specified as the *ClientInfoImpl* in the Deployment Descriptor web.xml.



You can, however, develop your own implementation for the *ClientInfo* interface. If you do, you need to store this change in the initialization parameter of the Deployment Descriptor.

(See also [The Deployment Descriptor \[Page 33\]](#)).

The following graphic shows the device recognition process schematically within the Java-enabled Web server.



3.4.2.12.2.5 ClientInfo Interface

Definition

The `ClientInfo` interface provides a range of methods for developing Java applications for mobile devices. These methods take into account the different display options for Web applications on different browsers, along with other device-specific properties – such as screen size or input method.

Use

In this way, you can use the methods of the `ClientInfo` interface when you create Java applications, and thus take the – often substantial – differences in the display of Web pages on different devices into account.

So that you can estimate the significance for your Web application of the methods described in the following table, some of them have been assigned a priority. For those device properties not assigned a priority, you can decide yourself whether or not you want to use them in your Web application.

Device properties can be categorized by their significance as follows:

Priority	Title	Description
1	Prerequisite	This property is prerequisite for running the mobile application at all. In general, these conditions are fulfilled by all “real” devices that fulfil the HTML or WAP standard. Emulators, however, do not fulfil these prerequisites. Thus this property need not be checked in the application. At most, you need a check program that ascertains whether or not an emulator is suitable for executing the application

Uniform Resource Name (URN)

2	Application does not work	If this property is not assigned a value, the application may crash on some devices – that is, may not be executable
3	Loss of information	If this property is not assigned a value, the application may not show all its information. For example, a loss of information may occur if the paragraph attribute for automatic line breaks is switched off (<p mode=nowrap>) and the paragraph (in a WML page) contains text that does not fit on one line. If the mobile device does not support the isMarqueeTextSupported [Page 33] property, it is possible that the end of the text is simply truncated. For this reason we advise you to think carefully before switching off the <i>automatic line break</i> property – even if doing so would allow you, for example, to make better use of a small screen on a mobile telephone.
4	Unacceptable display	If this property is not assigned a value, the application may be displayed in a way that does not do justice to the look and feel of the device
5	Inappropriate display	If this property is not assigned a value, the application may not be displayed at its best. However, the display will not actually disturb the user.

Similarly, some properties are assigned to the markup languages WML (Wireless Markup Language) and HTML (Hypertext Markup Language) – that is, you specify the markup language for which each method is relevant and thus how it can meaningfully be used. For example, the method `isFramesSupported` is only significant for HTML, since only HTML-enabled browsers can interpret frames. In general, it is HTML browsers on handheld devices like PDAs and pocket PCs, WAP-enabled mobile telephones with small screens cannot display frames, because of their screen interfaces. If the appropriate table column does not contain a value, the method can be used in both markup languages.

The following source code fragment shows how you can access the methods of the `ClientInfo` interface:

```
ClientInfoFactory factory = ClientInfoFactory.newInstance();
ClientInfo clientInfo = factory.newClientInfo();
clientInfo.load(request);
if(clientInfo.isTitleSupported())
// do sth if title is supported
...
else
// do sth if title is not supported
...

```



To obtain more information on using some of these methods in your Web application, use the links in the table.

Methods

Method	Signature	Description	Possible values	Content type	Prio.
getAccept [Page 33]	method <code>getAccept</code> returning value type <code>String</code>	Equivalent to the HTTP request-header „USER-AGENT“.			
<code>isAlertingSupported</code>	method <code>isAlertingSupported</code> returning value type <code>boolean</code>	Specifies whether or not the device supports messaging – for example, using SMS.	“true” or “false”		1

Uniform Resource Name (URN)

isAnchorSupported	method isAnchorSupported returning value type boolean	Specifies whether or not the <anchor> tag is supported	"true" or "false"	WML	1
isAnchorPrevSupported	method isAnchorPrevSupported returning value type boolean	Specifies whether or not the anchor tag supports a "Back" action	"true" or "false"	WML	
isAppLinksSupported	method isAppLinksSupported returning value type boolean	Specifies whether or not you can call a local application on a device using a special link	"true" or "false"	HTML	3
getAppLinkTypes	method getAppLinkTypes returning value type String	Specifies what type of application links the device supports		HTML	
isBackHardWired [Page 33]	method getBackHardWired returning value type boolean	Specifies whether or not you can execute the "Back" function using a fixed key without an associated tag	"true" or "false"	WML	4
getBackLabel [Page 33]	method getBackLabel returning value type boolean	Specifies whether or not a label attribute must be specified to display a label	"true" or "false"	WML	3
isBackToAnyUriSupported	method isBackToAnyUriSupported returning value type boolean	Specifies whether the <do type="prev"> tag can lead to any Web address	"true" or "false"	WML	1
isBigSupported	method isBigSupported returning value type boolean	Specifies whether or not text can be formatted as "large"	"true" or "false"	WML	
isBoldSupported	method isBoldSupported returning value type boolean	Specifies whether or not text can be formatted as "bold"	"true" or "false"	WML	
getBreakingSpace [Page 33]	method getBreakingSpace returning value type String	Returns the smallest character string for an empty space			
getBrowserCategory [Page 33]	method getBrowserCategory returning value type String	Returns the browser category	„unknown“ „pocketie“ „avantgo“ „imode“ „palm“ „wap“ „epoc“		
getBrowserName [Page 33]	method getBrowserName returning value type String	Returns the browser name	„unknown“ „Internet Explorer“ „Netscape Navigator“ „mobile“		

Uniform Resource Name (URN)

getBrowserOs [Page 33]	method getBrowserOs returning value type String	Returns the operating system running on the device	„unknown“ „HPUX“ „Linux“ „MacPPC“ „SunOS“ „Win32“ „mobile“		
getBrowserVersion	method getBrowserVersion returning value type String	Returns the browser version, for example 5.5			
isCacheEnabledByDefault	method isCacheEnabledByDefault returning value type boolean	Specifies whether or not cache memory on the browser is activated by default	“true” or “false”		
isCertificatesSupported	method isCertificatesSupported returning value type boolean	Specifies whether or not the device supports client certificates	“true” or “false”		
getCharHeight [Page 33]	method getCharHeight returning value type short	Returns the screen height in rows			4
getCharWidth [Page 33]	method getCharWidth returning value type short	Returns the screen width in characters			4
getColorDepth	method getColorDepth returning value type short	Returns the color depth, for example 256 colors		HTML	
isColorSupported	method isColorSupported returning value type boolean	Specifies whether or not the device has a color screen	“true” or “false”	HTML	
getContentType	method getContentType returning value type String	Returns the content type, for example HTML or WML			2
getContentTypeVersion	method getContentTypeVersion returning value type String	Returns the version of the content type – for example, 3.2 for HTML 3.2 or 1.1 for WML 1.1			
isCookiesSupported [Page 33]	method isCookiesSupported returning value type boolean	Specifies whether or not browser cookies are supported	“true” or “false”		1
isCssSupported [Page 33]	method isCssSupported returning value type boolean	Specifies whether or not the browser supports CSSs (Cascading Style Sheets).	“true” or “false”	HTML	4

Uniform Resource Name (URN)

getCssVersion	method getCssVersion returning value type String	Returns the CSS version		HTML	
getDefaultActionDesign [Page 33]	method getDefaultActionDe sign returning value type String	Returns the default design for user interface elements representing an action	„link“ „button“ „softkey“ „linkAndSoft key“	WML	3
getDefaultBlockSeparator [Page 33]	method getDefaultBlockSep arator returning value type String	Returns the default string used to separate paragraphs			5
getDefaultBullet [Page 33]	method getDefaultBullet returning value type String	Returns the default symbol used to indicate points in a list		WML	5
getDefaultFormStyle [Page 33]	method getDefaultFormStyl e returning value type String	Returns the default display type for input masks	„onePage“ „menu“ „wizard“	WML	
getDefaultMenuStyle [Page 33]	method getDefaultMenuStyl e returning value type String	Returns the default display type for menus	„selectionLis t“ „linkList“	WML	4
getDeviceCategory [Page 33]	method getDeviceCategory returning value type String	Returns the device category	„unknown“ „Phone“ „PDA“ „Voice“ „PC“		
getDeviceName [Page 33]	method getDeviceName returning value type String	Returns the device name Unique ID for a set of device properties			
isDomSupported	method isDomSupported returning value type boolean	Specifies whether or not the browser supports a Document Object Model (DOM)	„true“ or „false“	HTML	
getDomVersion	method getDomVersion returning value type String	Returns the Document Object Model (DOM) version supported		HTML	
isEmphasizedSupported	method isEmphasizedSupp orted returning value type boolean	Specifies whether or not text can be formatted as „highlighted“	„true“ or „false“	WML	5
isEmulator	method isEmulator returning value type boolean	Specifies whether or not the device is to be categorized as an emulator	„true“ or „false“		
getFieldsetLayout [Page 33]	method getFieldsetLayout returning	Specifies how input fields that belong together are to be laid out using the	„notSupporte d“ „ignored“	WML	3

Uniform Resource Name (URN)

	value type String	<fieldset> tag	„beneath“ „beneathWithIndent“ „sideBySide“		
isFieldsetTitleVisible [Page 33]	method isFieldsetTitleVisible returning value type boolean	Specifies whether or not the title attribute of the <fieldset> tag is used as label.	“true” or “false”	WML	3
isFontProportional	method isFontProportional returning value type boolean	Specifies whether or not the default font of the device is a proportional font	“true” or “false”		
getFormFactor [Page 33]	method getFormFactor returning value type String	Returns the device's screen format	„PC“ „VGA“ „halfVGA“ „mediumLandscape“ „mediumPortrait“ „phone“ „unknown“		3
isFormMenuSupported [Page 33]	method isFormMenuSupported returning value type boolean	Specifies whether or not the browser supports the technique of displaying a selection menu with the input mask	“true” or “false”	WML	4
isFramesSupported [Page 33]	method isFramesSupported returning value type boolean	Specifies whether or not frames are supported	“true” or “false”	HTML	2
getGrayLevel	method getGrayLevel returning value type short	Returns the number of shades of gray supported in grayscale pictures	“true” or “false”		
isHorzScrollingSupported	method isHorzScrollingSupported returning value type boolean	Specifies whether or not the device has a horizontal scroll bar	“true” or “false”		
isHrefWithParamsSupported [Page 33]	method isHrefWithParamsSupported returning value type boolean	Specifies whether or not an Href attribute in a link can contain URL parameters.	“true” or “false”	WML	2
isHttpGetSupported	method isHttpGetSupported returning value type boolean	Specifies whether or not the HTTP GET is supported	“true” or “false”	WML	1
isHttpPostSupported	method isHttpPostSupported returning value type boolean	Specifies whether or not the HTTP POST is supported	“true” or “false”	WML	1
isImageAlignmentSupported	method isImageAlignmentS	Specifies whether or not a graphic can be aligned (left,	“true” or “false”	WML	

Uniform Resource Name (URN)

	upported returning value type boolean	right, or centered)			
isImageLinksSupported [Page 33]	method isImageLinksSuppo rted returning value type boolean	Specifies whether or not a graphic can be used as link.	"true" or "false"		2
getImageSizeMax	method getImageSizeMax returning value type short	Returns the maximum memory size of a graphic		WML	
getInputFormatDate	method getInputFormatDat e returning value type String	Returns the character string used to specify the format for dates		WML	
getInputFormatNumeric	method getInputFormatNu meric returning value type String	Returns the formatting character string used to specify the format for numeric values		WML	
isInputMethodHandWriting	method isInputMethodHand Writing returning value type boolean	Specifies whether or not the device supports handwritten input	"true" or "false"		
isInputMethodKeyboard	method isInputMethodKeyb oard returning value type boolean	Specifies whether or not the device supports input using a keyboard	"true" or "false"		
isInputMethodKeypad	method isInputMethodKeyp ad returning value type boolean	Specifies whether or not the device supports input using a telephone keypad	"true" or "false"		
isInputMethodKeypadIntell	method isInputMethodKeyp adIntell returning value type boolean	Specifies whether or not the device supports input using T9 text input or similar	"true" or "false"		
isInputMethodVoice	method isInputMethodVoice returning value type boolean	Specifies whether or not the device supports speech input	"true" or "false"		
isInputShownWithCaption [Page 33]	method isInputShownWithC aption returning value type boolean	Specifies whether or not the browser uses the title attribute of the <input> tag is used as a label	"true" or "false"	WML	3
isItalicSupported	method isItalicSupported returning value type boolean	Specifies whether or not text can be formatted as "italic"	"true" or "false"	WML	5

Uniform Resource Name (URN)

isJavaSupported	method isJavaSupported returning value type boolean	Specifies whether or not the device supports the Java programming language	"true" or "false"	HTML	1
getJavaVersion	method getJavaVersion returning value type String	Returns the Java version supported		HTML	
isLinksSeparated [Page 33]	method isLinksSeparated returning value type boolean	Specifies whether or not two consecutive links are visibly separated	"true" or "false"	WML	4
getLinkDecoration [Page 33]	method getLinkDecoration returning value type String	Returns the delimiters that are automatically added to the link text by the device – for example [] or < >.		WML	
getLinkTextWidth [Page 33]	method getLinkTextWidth returning value type short	Returns the maximum number of characters that a link can have to fit into one line.		WML	3
isLocalImagesSupported	method isLocalImagesSupported returning value type boolean	Specifies whether or not the device supports locally stored graphics or symbols	"true" or "false"		3
isLocalVariablesSupported	method isLocalVariablesSupported returning value type boolean	Specifies whether or not the browser supports local variables	"true" or "false"	WML	1
isMarqueeLinkSupported [Page 33]	method isMarqueeLinkSupported returning value type boolean	Specifies whether or not a long link can be displayed in one line, for example as a ticker tape.	"true" or "false"	WML	3
isMarqueeTextSupported [Page 33]	method isMarqueeTextSupported returning value type boolean	Specifies whether or not a long text can be displayed in one line, for example as a ticker tape.	"true" or "false"	WML	3
getMaxLinkLength [Page 33]	method getMaxLinkLength returning value type short	Returns the maximum size of the Href attributes of a link.			1
getMediaFormats [Page 33]	method getMediaFormats returning value type String	Returns the list of multimedia formats supported – such as .agif (animated .gif format), .gif, .jpg, .png, .wbmp			3
getMemory	method getMemory returning value type String	Returns the maximum memory capacity of the device		HTML	
getModel	method getModel returning	Returns the name of the device type – such as 7110			

Uniform Resource Name (URN)

	value type String	for the Nokia 7110 mobile telephone			
isNbspSupported [Page 33]	method isNbspSupported returning value type boolean	Specifies whether or not the device supports non-breaking blank characters	"true" or "false"	WML	4
isNestedTablesSupported [Page 33]	method isNestedTablesSupported returning value type boolean	Specifies whether or not the browser supports nested tables	"true" or "false"	HTML	2
isNewlineAfterImage [Page 33]	method getNewlineAfterImage returning value type boolean	Specifies whether or not a line break is inserted automatically after an tag.	"true" or "false"	WML	4
isNewlineAfterInput [Page 33]	method isNewlineAfterInput returning value type boolean	Specifies whether or not a line break is inserted automatically after an <input> tag on a specific mobile device.	"true" or "false"	WML	4
isNewlineAfterLink	method getNewlineAfterLink returning value type boolean	Specifies whether or not a line break is inserted automatically after a link	"true" or "false"	WML	4
isNewlineAfterSelect	method isNewlineAfterSelect returning value type boolean	Specifies whether or not a line break is inserted automatically after an <select> tag.	"true" or "false"	WML	4
isNewlineBeforeImage [Page 33]	method isNewlineBeforeImage returning value type boolean	Specifies whether or not a line break is inserted automatically before an tag.	"true" or "false"	WML	4
isNewlineBeforeInput [Page 33]	method isNewlineBeforeInput returning value type boolean	Specifies whether or not a line break is inserted automatically before an <input> tag.	"true" or "false"	WML	4
isNewlineBeforeLink [Page 33]	method isNewlineBeforeLink returning value type boolean	Specifies whether or not a line break is inserted automatically before a link	"true" or "false"	WML	4
isNewlineBeforeSelect	method isNewlineBeforeSelect returning value type boolean	Specifies whether or not a line break is inserted automatically before an <select> tag.	"true" or "false"	WML	4
isNewlineBetweenImages [Page 33]	method isNewlineBetweenImages returning	Specifies whether or not a line break is inserted automatically after an <image> tag	"true" or "false"	WML	3

Uniform Resource Name (URN)

	value type boolean				
isNewlineBetweenLinks [Page 33]	method isNewlineBetweenLinks returning value type boolean	Specifies whether or not a line break is inserted automatically between two <image> tags	"true" or "false"	WML	4
isNewlineBetwLinkAndTag [Page 33]	method isNewlineBetwLinkAndTag returning value type boolean	Specifies whether or not a line break is inserted automatically between a link and a tag	"true" or "false"	WML	4
isOfflineBrowsingSupported	method isOfflineBrowsingSupported returning value type boolean	Specifies whether or not the browser supports offline browsing through locally stored pages (that is, cached pages)	"true" or "false"	HTML	2
isOfflineFormsSupported	method isOfflineFormsSupported returning value type boolean	Specifies whether or not the browser allows the user to fill out input forms offline on the device	"true" or "false"	HTML	2
getPageSizeMax [Page 33]	method getPageSizeMax returning value type int	Returns the maximum page size that can be processed in a mobile device.			2
getPixelHeight [Page 33]	method getPixelHeight returning value type short	Specifies the screen height in pixels.			
getPixelWidth [Page 33]	method getPixelWidth returning value type short	Specifies the screen width in pixels.			
isRedirAbsoluteSupported	method isRedirAbsoluteSupported returning value type boolean	Specifies whether or not the browser supports the redirection of an absolute URL address.	"true" or "false"		1
isRedirRelativeSupported [Page 33]	method isRedirRelativeSupported returning value type boolean	Specifies whether or not the browser supports the redirection of a relative URL address.	"true" or "false"		2
isScriptSupported [Page 33]	method isScriptSupported returning value type boolean	Specifies whether or not the browser supports scripting	"true" or "false"		2
getScriptVersion	method getScriptVersion returning value type String	Returns the script version supported			
isSecureProtocolsSupported	method isSecureProtocolsSupported returning value type boolean	Specifies whether or not the browser supports SSL (Secure Socket Layer) or WTLS (Wireless Transport Layer Security)	"true" or "false"		

Uniform Resource Name (URN)

getSecureProtocolNames	method getSecureProtocols Names returning value type String	Returns the names of the security protocols supported The names are separated by semicolons			
isSelectionMenuSupported [Page 33]	method isSelectionMenuSupported returning value type boolean	Specifies whether or not the menu layout type "selectionList" [External] is supported	"true" or "false"	WML	4
isSetvarOnEventSupported	method isSetvarOnEventSupported returning value type boolean	Specifies whether or not the <setvar> tag can be used within the <onevent type="onenterforward"> event handler	"true" or "false"	WML	1
isSkippingToInput [Page 33]	method isSkippingToInput returning value type boolean	Specifies whether or not the browser automatically skips to the first <input> tag and displays a screen extract around this tag	"true" or "false"	WML	4
isSmallSupported	method isSmallSupported returning value type boolean	Specifies whether or not text can be formatted as "small"	"true" or "false"	WML	5
getSoftkeyNum [Page 33]	method getSoftkeyNum returning value type short	Returns the number of soft keys supported by the device		WML	3
getSoftkeyStyle1 [Page 33]	method getSoftkeyStyle1 returning value type String	Specifies how soft key 1 is displayed on the screen	„notShown“ „key“ „menu“ „screen“	WML	5
getSoftkeyStyle2 [Page 33]	method getSoftkeyStyle2 returning value type String	Specifies how soft key 2 is displayed on the screen	„notShown“ „key“ „menu“ „screen“	WML	5
getSoftkeyTitleWidth [Page 33]	method getSoftkeyTitleWidth returning value type String	Returns the number of displayed characters for a soft key title.		WML	3
isSoundSupported	method isSoundSupported returning value type boolean	Specifies whether or not the device can play sounds	"true" or "false"	HTML	
isStrongSupported	method isStrongSupported returning value type boolean	Specifies whether or not text with the can be formatted as "highlighted"	"true" or "false"	WML	5
isSubmitOneventSupported	method isSubmitOneventSupported returning value type boolean	Specifies whether or not "Submit" is supported within the <onevent type="onenterforward"> event handler	"true" or "false"		1
getSubCategory [Page 33]	method getSubCategory	Allows you to split devices	Any text		

Uniform Resource Name (URN)

	returning value type String	into different sub-categories			
isTableHasBorders [Page 33]	method isTableHasBorders returning value type boolean	Specifies whether or not tables are displayed with gridlines	"true" or "false"		4
isTableSupported [Page 33]	method isTableSupported returning value type boolean	Specifies whether or not the browser supports tables with several columns	"true" or "false"		3
isTelephonyLinksSupported [Page 33]	method isTelephonyLinksSupported returning value type boolean	Specifies whether or not a telephone can be dialed directly using a link	"true" or "false"		4
isTextAlignmentSupported	method isTextAlignmentSupported returning value type boolean	Specifies whether or not text within a paragraph can be aligned left, right, or centered	"true" or "false"	WML	5
isTextStylesSupported [Page 33]	method isTextStylesSupported returning value type boolean	Specifies whether or not the browser can format text using tags such as or <small>	"true" or "false"	WML	5
isTitleSupported [Page 33]	method isTitleSupported returning value type boolean	Specifies whether or not a label is to appear on the top of the screen using the "title" property of the <card> WML tag.	"true" or "false"	WML	3
getTitleWidth [Page 33]	method getTitleWidth returning value type short	Returns the maximum number of characters of the title			3
isUnderlineSupported	method isUnderlineSupported returning value type boolean	Specifies whether or not text can be formatted as "underlined"	"true" or "false"	WML	5
getUserAgent [Page 33]	method getUserAgent returning value type String	Equivalent to the HTTP request header "userAgent"			
isVarsAcrossCardSupported	method isVarsAcrossCardSupported returning value type boolean	Specifies whether or not browser variables passed to a card can also be used for different cards	"true" or "false"	WML	1
getVendor	method getVendor returning value type String	Returns the manufacturer's name			
isXslSupported	method isXslSupported returning value type boolean	Specifies whether or not the browser supports Extensible Stylesheet Language (XSL)	"true" or "false"	HTML	

Uniform Resource Name (URN)

getXslVersion	method getXslVersion returning value type String	Returns the XSL version supported		HTML	
---------------	---	-----------------------------------	--	------	--

3.4.2.12.2.5.1 getAccept Method**Use**

The getAccept method returns a value that is equivalent to the value of the HTTP request header ACCEPT. This header is sent to the Web server by a device in an HTTP request and contains the complete information about the [MIME-Types \[External\]](#) (or Content types) that can be processed by the browser.

The following list contains MIME types that a browser can pass in a request:

MIME Type	Description	File Extension
text/html	HTML file	html, htm
text/css	CSS style sheet file	css
text/javascript	JavaScript file	js
image/gif	GIF graphics	gif
image/jpeg	JPEG graphics	jpeg, jpg, jpe
application/msword	Microsoft Word files	doc, dot

The MIME types based on WAP include:

MIME Type	Description	File Extension
text/vnd.wap.wml	WML source code	wml
application/vnd.wap.wmlc	Compiled WML	wmlc
text/vnd.wap.wmlscript	WMLScript	wmlscript
text/vnd.wap.wmlscript	WMLScript	wsc
application/vnd.wap.wmlscriptc	Compiled WMLScript	wmlsc
application/vnd.wap.wmlscriptc	Compiled WMLScript	wsc
image/vnd.wap.wbmp	Wireless Bitmap	wbmp

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.2 isBackHardWired Method

Use

The `isBackHardWired` method provides information on whether the *Back* function, which calls the last page visited, can be executed with a fixed key or has to be defined and programmed with a soft key.

The individual device types provide the user with varyingly customized keys for operating the application. Not all WAP-enabled devices have a fixed key for the *Back* function.

With a soft key, which is displayed by a label in the menu bar at the bottom of the screen, you can assign a function to a key. The function is assigned to the key using the WML tag `<do>` and executed by pressing this key.

The WML code for assigning the *Back* function to a key is:

```
<do type="prev" label="Back">
  <prev/>
</do>.
```

The `isBackHardWired` method returns a Boolean value. The value for *true* means that the *Back* function is assigned to a key. In this case, the `<do>` tag is not required.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.3 getBackLabel Method

Use

The `getBackLabel` method of the `ClientInfo` interface specifies whether or not a mobile device is to appear on the top of the screen using the label attribute of the `<do>` WML tag. (See also [isBackHardWired Method \[Page 33\]](#)).

If the label attribute is not used for the label, you should not assign a label attribute to the `<do>` tag or you should use another type (for example, `type="options"`) or a link.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.4 getBreakingSpace Method

Use

The `getBreakingSpace` method returns the character string that can be used, for example, for the separation of two links that otherwise would appear successively.



Unlike a forced blank character (` `) this character string enables the browser to add a line break between the links if necessary.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.5 **getBrowserCategory Method**

Use

The values of this method are a rough categorization of the browser variants.



If you have already created mobile applications using the Internet Transaction Server (ITS), you should be advised that the values provided by the getBrowserCategory method correspond to the values of the context variable ~markupsbtype that are returned by the device recognition.

Values

Values	Short Description
unknown	Unknown browser type
pocketie	Pocket Internet Explorer
avantgo	AvantGo HTML 3.2 Browser
imode	Micro browser for I-mode devices
palm	HTML browser for Palm devices
wap	Micro browser for WAP-enabled devices
epoc	HTML browser for EPOC devices

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.6 **getBrowserName Method**

Use

A detailed value definition for the getBrowserName method is not useful due to the rapid growth of the mobile device market.

This is a free text field pre-assigned with reliable values by SAP.



However, the system administrator can change these entries for a specific device.

Values

Values	Short Description
unknown	Unknown browser
Internet Explorer	Microsoft Internet Explorer
Netscape Navigator	Netscape Navigator
mobile	Browser on the mobile device

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.7 **getBrowserOS Method**

Use

The getBrowserOs method returns the operating system of the requesting device.

Values

Values	Short Description
unknown	Unknown operating system
HPUX	HP Unix operating system
Linux	Linux operating system
MacPPC	Macintosh Power PC operating system
SunOS	Sun Solaris operating system
Win32	Microsoft Windows 32 bit operating system
mobile	Operating system of a mobile device

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.8 **getCharHeight Method**

Use

The getCharHeight method returns the maximum number of lines that can be displayed on the screen.



Web applications are more user-friendly if information that belongs together fits onto one screen and there is no need to browse between the individual pages. Using the getCharHeight method the development team can define, for each specific device, the maximum number of lines that can be displayed on the screen for each document.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.9 **getCharWidth Method**

Use

In case the mobile devices that do not support proportional fonts and therefore the width of each character is the same, the getCharWidth method returns the maximum number of characters that can be displayed in a screen line.

Uniform Resource Name (URN)

This number might differ from the actual character number for a line when using proportional fonts. In this case the value that is returned by the `getCharWidth` method is the maximum number of characters with an average character width.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.10 isCookiesSupported Method

Use

The `isCookiesSupported` method specifies whether or not the browser or its Gateway supports cookies.

[Cookies \[External\]](#) are especially used for session administration. If they are not supported by the mobile device, the application must be modified accordingly, otherwise errors might occur.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.11 isCssSupported Method

Use

The `isCssSupported` method specifies whether or not the browser supports CSS (Cascading Style Sheet).

If an application uses Cascading Style Sheets, but the browser of the mobile device does not support this function, the application cannot be displayed properly because the style sheets are not evaluated. Colors and fonts are missing.

However, you can display the CSS layout if you directly specify the rendering attributes of the Cascading Style Sheets as HTML attributes.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.12 getDefaultActionDesign Method

Use

The `getDefaultActionDesign` method specifies which preferred type of interface element is to be used for the display of actions.

The following are actions:

- Navigation steps (links)
- Processing steps that affect the current screen (for example save, delete, scroll)
- Initiation of device-specific actions, for example phone calls or starting an application on the mobile device.

These interface elements include buttons, links and soft keys. The latter are specifically for WAP-enabled devices. The selection of a specific interface element for executing functions depends on the requesting mobile device. For HTML markup, mainly buttons and links are

Uniform Resource Name (URN)

used to execute these functions, while links and soft keys are normally used for WAP-enabled devices.

Values

Values	Short Description
link	The action appears as a link.
button	The action can be triggered by choosing a button.
softkey	The action appears as soft key.
linkAndSoftkey	The action appears as a link or a soft key.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.13 getDefaultBlockSeparator Method

Use

The getDefaultBlockSeparator method returns a default value for a separator between paragraphs. The end of a page often features such a separator. The default separator is the character string “---”.

Using this method has the following advantages over the text definition of the list character:

- Optimized separators are used for each device.
- The separators are consistent in all applications.
- The separators can be changed centrally by the system administrator without having to modify applications.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.14 getDefaultBullet Method

Use

The getDefaultBullet method returns a default value for a list character that is to be used in lists, for example. For most devices, the value is set to the character » (code in text »). For mobile devices that cannot display this character at all or appropriately, another character is used, for example a dash.

Using this method has the following advantages over the text definition of the list character:

- The list character optimized for each device is used.
- The list characters are consistent in all applications.
- The list characters can be changed centrally by the system administrator without having to modify applications.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.15 getDefaultFormStyle Method

Use

The getDefaultFormStyle method specifies which preferred type of input template is to be used for a specific mobile device.

Features

The <input> and <select> tags enable you to use input templates for customizing the interaction with the user and to allow data entry by the user.

Two options are available for data input:

- Input for optional text input by using the <input> tag
- Selection lists with check boxes and option groups by using the <select> tag

The display of these tags on the screen of a mobile device can vary, especially if a card contains several input fields that cannot be displayed on the screen simultaneously.

In some devices, the user can scroll on the page to reach input fields that are not immediately visible. However, after filling the input fields, some browsers display the input fields on different screens and distribute the screen contents over different pages.

Values

Values	Short Description
onPage	The input fields are displayed one after another and are visible on the same screen.
menu	The input fields and selection lists are displayed in one overview page. To fill the fields an additional page is called – one page for each input field or selection option.
wizard	The input fields and selection lists are displayed so that each field has a card.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.16 getDefaultMenuStyle Method

Use

The getDefaultMenuStyle method specifies which preferred type of menu layout is to be used for a specific mobile device. The device properties delivered by SAP use the values *linkList* and *selectionList*, because there are two established variants for WAP-enabled devices:

- Numbered lists of options. This variant is mainly used on WAP-enabled devices with Openwave browser. The various menu options are implemented as <option> tags within a <select> tag. The Openwave browser displays this as a numbered list and a menu option can be selected by navigating or using the appropriate number key.

Uniform Resource Name (URN)



It is strongly recommended to use this variant on Openwave browsers, because otherwise there is a significant difference in look-and-feel to other applications. However, on most of the other browsers this variant is not useful, because the <select> and <option> tags do not implement a menu-like display.

- **Link list.** The various menu options are displayed as links below one another or side by side. To select a menu option the user must navigate to the link and select it. As usual, these links are implemented by using the <a> or <anchor> tag. This variant is universal and can be used on all browsers, but for Openwave browsers you should use the first implementation variant for the reasons mentioned above.

Values

Values	Short Description
selectionList	The menu is displayed as a numbered list, and the menu options are selected directly by using the number keys of the mobile telephone.
linkList	The menu is displayed as a link list. The user must navigate to the desired link and select it.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.17 **getDeviceCategory Method**

Use

The values that are returned by the getDeviceCategory method represent a rough categorization of the device types.

Values

Values	Short Description
unknown	Unknown device type
Phone	Data-driven mobile telephone or Smartphone
PDA	Mobile device such as Pocket PC, Palm or hand-helds
Voice	Device controlled by voice input
PC	Complete personal computer

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.18 **getDeviceName Method**

Use

The getDeviceName method is used to uniquely identify a device property set. The value that is returned by the getDeviceName method is the key of the table, whose data returns the values of the properties of the requesting mobile device. Only one browser type is assigned to the name of devices that offer several browser types.



Because the manufacturers of mobile devices are constantly launching new products, it is possible that no suitable device is found for an HTTP request in the database. Since the getDeviceName method must return a unique value, generic values are entered for unknown device types.

This method is used when an application is to be customized for a specific device.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.19 **getFieldsetLayout Method**

Use

The getFieldsetLayout method specifies the assignment of several related input fields using the <fieldset> tag.

Features

The <fieldset> tag is used to group related input fields in a Web application. This allows the browser to optimize the display and navigation and display specific contents on the screen simultaneously.

Values

Values	Short Description
notSupported	<fieldset> tag is not supported by the device.
ignored	<fieldset> tag is ignored, the input fields are displayed below one another.
beneath	The input field groups are displayed below one another.
beneathWithIndent	The input field groups are displayed below one another, and the input fields are indented.
sideBySide	The input field groups are displayed side by side.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.20 isFieldsetTitleVisible Method

Use

The isFieldsetTitleVisible method specifies whether or not the title attribute of the <fieldset> tag is used as label.

The <fieldset> tag is used to group related input fields in a Web application. If the <fieldset> tag can represent the title, this feature should be used because it improves the page structure on some devices. If not, the title should be displayed as normal text.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.21 getFormFactor Method

Use

The getFormFactor method returns the form factor (screen format) of the mobile device.

Features

An important feature for customizing a Web application is the form factor (screen format) of the mobile device on which the application is to be used. Form factor is the physical size and shape of a mobile device. The transitions regarding shape and form of devices can be seamless, and therefore the categorization of exact screen measures for mobile devices is not useful.

The device recognition process returns a value that helps to decide how to build the graphical user interface of a Web application.

Values

The form factor that is returned by the getFormFactor method provides a rough categorization (recommended by SAP) with the following values:

Values	Short Description	Example
PC	Large screen in full VGA format or greater	
VGA	Screen in full VGA format	
halfVGA	Screen in half VGA format	
mediumLandscape	Medium screen size as used by Smartphones	For example 640 x 200 pixel for Nokia Communicator
mediumPortrait	Medium screen size as used by PDA's (Personal Digital Assistant [External])	For example 320 x 240 pixel for Pocket PC 160x160 pixel Palm
phone	Mobile telephone screen	
unknown	Unknown device	

Example

Uniform Resource Name (URN)

You can use this attribute for displaying data in a table. For the mediumPortrait format, which specifies the form factor of a [PDA \[External\]](#), it is useful to display data in tables, but other options normally have to be chosen for mobile telephones due to their small screen size.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.22 isFormMenuSupported Method

Use

The isFormMenuSupported method specifies whether, using the attribute *ordered="false"* in the <card> tag, the browser can be made to place a page with a selection menu before a page with input fields.

In this selection menu, the input fields are displayed in a numbered list. This enables the user to go directly to an input field by entering the relevant number.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.23 isFramesSupported Method

Use

The isFramesSupported method specifies whether or not the browser supports frames.

If the device cannot display frames, the application must be modified, because it cannot be displayed correctly.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.24 isHrefWithParamsSupported Method

Use

The isHrefWithParamsSupported method specifies whether or not an Href attribute in a link can contain URL parameters.



If a device does not support this, parameters should not be used, since errors might occur.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.25 isInputShownWithCaption Method

Use

The `getInputShownWithCaption` method specifies the various display types of the *title* attribute of the `<input>` tag on different WAP-enabled devices.

With some devices, Ericsson devices for example, the value of the title attribute appears as a label before the input field. In this case, you should not use another label before the input field.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.26 isImageLinksSupported Method

Use

The `isImageLinksSupported` method specifies whether or not a graphic can be used as link.



If this is not the case, a normal link should be used, otherwise the functions of the page are restricted.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.27 isLinksSeparated Method

Use

The `isLinksSeparated` method specifies whether or not links are to be displayed separately. This optical separation can be caused by:

- A line break between links
- Delimiting a link, for example with [] or < >
- Automatic insertion of a blank character.

If links cannot be separated, a blank character should be inserted to distinguish the individual links.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.28 getLinkDecoration Method

Use

The `getLinkDecoration` method returns the delimiters that are automatically added to the link text by the device. Some mobile devices highlight the links by using square or angle brackets as delimiters. The layout for the same browser may vary on different devices.

Uniform Resource Name (URN)

These additional characters reduce the space that is available for information. This method helps to calculate the available space.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.29 **getLinkTextWidth Method**

Use

The getLinkTextWidth method returns the maximum number of characters that a link can have to still fit into one line.



For small WAP-enabled devices, the length of the link text should not exceed 14 characters. If you want to offer longer link texts for larger devices, you should define different variants dependent on this attribute.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.30 **isMarqueeLinkSupported Method**

Use

The isMarqueeLinkSupported method specifies whether or not a long link can be displayed in one line, for example as a ticker tape.

For small devices that support this attribute, you can also use long link texts. (See also [getLinkTextWidth Method \[Page 33\]](#))

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.31 **isMarqueeTextSupported Method**

Use

The isMarqueeTextSupported method specifies whether or not a long text can be displayed in one line, for example as a ticker tape.



If you want to display a long text in one line instead of having a line break, you can use the enclosing tag `<p nowrap="true">`. However, you should only use it if this value is met. Otherwise the text may be truncated.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.32 **getMaxLinkLength Method**

Use

The getMaxLinkLength method returns the maximum size of the Href attributes of a link.

The maximum value of this attribute is 256, regardless of the actual value. Your links should not exceed this size, since otherwise the application will not function correctly.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.33 **getMediaFormats Method**

Use

The getMediaFormats method returns the list of supported multimedia formats (separated by semicolons).

You must be able to evaluate the supported multimedia formats in an application to avoid the use of graphics that cannot be displayed. Most WAP browsers only support the WBMP format; while HTML browsers usually support GIF and JPEG.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.34 **isNbspSupported Method**

Use

The isNbspSupported method specifies whether or not the device supports non-breaking blank characters.



If a mobile device does not support this property, the expression cannot be used appropriately for text formatting.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.35 **isNestedTablesSupported Method**

Use

The isNestedTablesSupported method specifies whether or not the browser supports nested tables.

Since the WAP standard does not support nested tables, this attribute is only relevant for HTML browsers. Some of these browsers do not allow nested tables either.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.36 isNewlineAfterImage Method

Use

The `isNewlineAfterImage` method specifies whether or not a line break is inserted automatically after an `` tag. If a mobile device displays two `` tags side by side, a `
` tag must be inserted to display in a new line. In mobile devices that meet the condition, only the `isNewlineAfterImage` method can prevent the screens from being moved apart by inserted blank lines.



The same arguments for the use of this method apply to all methods that concern the line break into a new line.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.37 isNewlineAfterInput Method

Use

The `isNewlineAfterInput` method specifies whether or not a line break is inserted after an `<input>` tag or the subsequent tag is displayed in the same line. If this method returns the value *true*, the subsequent tag is displayed in the next line. In this case, a `
` tag could insert an unwanted blank line.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.38 isNewlineBeforeImage Method

Use

The `isNewlineBeforeImage` method specifies whether or not a line break is inserted automatically before an `` tag.

If you want graphics to start in a new line, you must use the `
` tag in devices that return the value *false*. In mobile devices that meet this condition, this could result in the insertion of a blank line, which you want to avoid due to space restrictions on small screens.



The same arguments for the use of this method apply to all methods that concern line breaks into new lines.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.39 isNewlineBeforeInput Method

Use

The isNewlineBeforeInput method specifies whether or not a line break is inserted automatically before an <input> tag.

If you want the <input> tag to appear in the following line of a Web application, you must use the
 tag for the line break. A mobile device that automatically inserts a line break before an <input> tag and therefore returns the value *true*, would then display an unwanted blank line.



The same arguments for the use of this method apply to all methods that concern line breaks into new lines.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.40 isNewlineBeforeLink Method

Use

The isNewlineBeforeLink method specifies whether or not a line break is inserted automatically before a <link> tag.

If a mobile device meets this condition, the link cannot be displayed in the line of the preceding text.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.41 isNewlineBetweenImages Method

Use

The isNewlineBetweenImages method specifies whether or not a line break is inserted automatically between two tags.



If a mobile device meets this condition, two successive screens cannot be displayed in the same line, even if so desired.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.42 isNewlineBetweenLinks Method

Use

The isNewlineBetweenLinks method specifies whether or not a line break is inserted automatically between two links.



If a mobile device meets this condition, the link cannot be displayed in the same line as the previous link, even if so desired.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.43 isNewlineBetwLinkAndTag Method

Use

The isNewlineBetwLinkAndTag method specifies whether or not a line break is inserted automatically between a link and a tag. Some mobile devices display a text before a link in a new line if this text is, for example, enclosed by a font tag such as .



If a mobile device meets this condition, a link that follows a formatted text is always displayed in a new line, even if this is not wanted.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.44 getPageSizeMax Method

Use

The getPageMax method returns the maximum page size that can be processed on a mobile device.



In [WAP \[External\]](#) applications, this corresponds to the size of the compiled WML of a deck in bytes. Each WML page, known as a deck, can be divided into different cards that you can switch between using links.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.45 getPixelHeight Method

Use

The getPixelHeight method specifies the screen height in pixels.

This method can be used if, for example, you want to integrate screens of different sizes into the application. For mobile devices with a bigger screen format (form factor), the getPixelHeight method enables you to load bigger and therefore more legible graphics into the user interface.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.46 **getPixelWidth Method**

Use

The getPixelWidth method specifies the screen width in pixels.

Similar to the [getPixelHeight Method \[Page 33\]](#) this method specifies which screen width can be used with a specific mobile device.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.47 **isRedirRelativeSupported Method**

Use

The isRedirRelativeSupported method specifies whether or not the browser supports the rerouting of a relative URL address.



If you want to redirect the browser to a different URL, you can use the *Redirect* function. However, some WAP browsers do not support this function for relative addresses (URLs). If you disregard this browser feature, the application, in which the *Redirect* function is used, will not work.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.48 **isScriptSupported Method**

Use

The isScriptSupported method specifies whether or not the browser supports scripting.

Scripting allows you to improve the layout of many pages and add client-side functions. Be aware that on browsers that do not support scripting, these pages may become unusable.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.49 **isSelectionMenuSupported Method**

Use

The isSelectionMenuSupported method specifies whether or not the WAP browser displays a <select> tag as a numbered list of selection values.

If a mobile device meets this condition, you can use the <select> tag to implement a selection menu; the selection is made by pressing a number key.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.50 isSkippingToInput Method

Use

The isSkippingToInput method specifies whether or not the browser automatically skips to the first <input> tag and displays a screen extract around this tag.



If the <input> tag in a Web application appears towards the end of an extensive page, the information of the Web application that appears at the beginning may not be readable immediately and important data might not be available. To find out what data needs to be entered in the input field, the user must scroll back to the location of the information. This makes the Web application awkward to use and also makes it difficult to navigate in the pages.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.51 getSoftkeyNum Method

Use

The getSoftkeyNum method returns the number of available soft keys that can be created with <do> tags.



When defining a page layout, you must ensure that you do not assign important functions to more soft keys than are available.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.52 getSoftkeyStyle1 Method

Use

The getSoftkeyStyle1 method specifies how the type attribute *accept* of the <do> tag is displayed on the mobile device screen.

Features

The <do> tag is used to execute actions either by pressing a device key or by choosing a menu option.

Therefore, the <do> tag enables you to customize additional navigation and control options in addition to navigation with links. For this purpose, you can access the predefined functions (such as go, prev, or noop) using the type attribute in the <do> tag. For example, you can switch to the next card by using the <go> action. This is why the <do> tag plays an important role in navigation on mobile devices.

The display of the <do> tag on the screen is browser-dependent and the tag can be presented graphically as a soft key in the menu bar at the bottom of the screen.

(see also [isBackHardWired Method \[Page 33\]](#) and [getSoftkeyStyle2 Method \[Page 33\]](#))

Uniform Resource Name (URN)

A soft key is similar to a control or an interface element. You can use it to execute processes if the <do> event occurs after pressing a device key.

You specify the type of the event using the type attribute of the <do> tag. This means you pass information about the content of the event and the associated action to the mobile device. The mobile devices use this information to find an appropriate display type for the execution of this action.

If you assign the value *accept* to the type attribute of the <do> tag, interface elements are displayed that can be used for confirmation, for example, of a selection.



On some devices, the <do> tag is displayed as a soft key at the bottom of the screen and the corresponding action is executed with a device key. This type of implementation of the <do> tag on mobile devices is very common, however, on some devices, the <do> tag is displayed as a selection option in the menu, as a button on a touch screen of a mobile hand-held device, or it is not displayed at all.

Values

Values	Short Description	Example
notShown	The <do> tag does not appear on the screen.	
key	The <do> tag is displayed as a soft key and the function is assigned to a device key.	
menu	The <do> tag appears as a selection option in the menu.	
screen	The <do> tag appears as a button on the screen.	For example, on the touch screen of a hand-held device like a PDA [External] or Pocket PC. Tapping the screen (with a special pen) triggers the action.

Example

The following WML code is an example for the use of soft keys in a WML card.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Card1" title="Card1">
<do type="accept" label="Next">
<go href="#Card2"/>
</do>
<p>
Choose Next to continue
</p>
</card>
<card id="Card2" title="Card2">
<p>
Card2
</p>
```

```
</card>  
</wml>
```

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.53 getSoftkeyStyle2 Method

Use

The getSoftkeyStyle2 method specifies the display of the type attribute *options* of the <do> tags on the screen of the mobile device.

Features

Using the <do> tag offers the user more navigation and control options on mobile devices, and for some devices enables you, for example, to customize the menu bar at the bottom of the screen. You can use predefined actions like go, prev, help, or noop as controls. These are executed by pressing the corresponding keys on the mobile device.

The display of this tag on the screen is browser-dependent and in some devices the tag can be converted into a visible soft key, a kind of navigation button, in the menu bar at the bottom of the screen. For example, you can switch to the next card by using the <go> action and then directly select a Web address by using the interface element that is defined by the <do> tag. In so doing, the <do> tag assigns a defined function to a key on the mobile device. If you press this key, the <do> event occurs, which in turn triggers the <go> action.

You specify the type of the event using the type attribute of the <do> tag. This means you pass information about the content of the event and the associated action to the mobile device. The mobile devices use this information to find an appropriate display type for the execution of this action.

If you assign the “options” value to the type attribute of the <do> tag, interface elements are displayed that you can use to call a list of context-sensitive options or a selection list.



The type value “options” should be used in a <do> tag if you have already defined a soft key of the type “accept” in a card.

On some devices, the <do> tag is displayed as a soft key at the bottom of the screen and the corresponding action is executed with a device key. This type of implementation of the <do> tag on mobile devices is very common, however, on some devices, the <do> tag is displayed as a selection option in the menu, as a button on a touch screen of a mobile hand-held device, or it is not displayed at all.

(see also [getSoftkeyStyle1 Method \[Page 33\]](#) and [isBackHardWired Method \[Page 33\]](#))

Values

Values	Short Description	Example
notShown	The <do> tag does not appear on the screen.	
key	The <do> tag is displayed as a soft key and the function is	

	assigned to a device key.	
menu	The <do> tag appears as a selection option in the menu.	
screen	The <do> tag appears as a button on the screen.	For example, on the touch screen of a hand-held device like a PDA [External] or Pocket PC. Tapping the screen (with a special pen) triggers the action.

Example

The following WML code is an example for the use of soft keys.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card title="Page1">
<do type="accept" label="Continue">
<go href="page2.wml"/>
</do>
<do type="options" label="Exit">
<exit/>
</do>
</p>...
</card>
</wml>
```

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.54 getSoftkeyTitleWidth Method

Use

The getSoftkeyTitleWidth method returns the number of displayed characters for a soft key title.



For a small WAP-enabled device, the length of the soft key label should not exceed 6 characters. You should define different variants of this property if you want to offer longer label texts for specific devices.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.55 getSubCategory Method

Use

The getSubCategory method is used for the refinement of the device categorization.

(See also [getDeviceCategory Method \[Page 33\]](#)).

However, SAP does not predefine values. Therefore the system administrator can implement a company-specific categorization of the mobile devices.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.56 isTableHasBorders Method

Use

The isTableHasBorders method specifies whether or not tables are displayed with gridlines.



If a mobile device meets this condition, the `<table>` tag cannot be used for the layout of forms in your application. This is especially the case if input fields and buttons are to be arranged form-like and justified. Only genuine tabular arrangements should use the table. However, this can also cause problems on mobile browsers that do not display the table attributes cellspacing and cellpadding as desired.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.57 isTableSupported Method

Use

The isTableSupported method specifies whether or not a mobile device can interpret the `<table>` tag, so multiple columns can be displayed side by side.

Features

Tables are a crucial element for the clear layout of Web applications. It is essential that you define the number of columns in a table using the tag attribute *columns*.

For example, there are mobile devices that can only display one column on each page. So, for example, WAP-enabled Nokia devices do not interpret tables in the usual way. For these devices, the value is set to *false*, because the contents of the columns are not displayed side by side but below one another. The layout becomes too complex and you can no longer correctly assign the column values.

However, an Openwave browser displays the tables in the usual way – that is, it displays columns side by side.

If a mobile device does not meet the condition, because it does not support the `<table>` tag, another display type for the structure of the Web application should be selected (see example). For some devices, you can use lists instead of tables.

Example

The following JSP example shows one way of how to use the isTableSupported method for the customization of Web applications.

```
<%@ page import="com.sap.mobile.clientinfo.*" %>
<%
ClientInfoFactory factory = ClientInfoFactory.newInstance();
ClientInfo clientInfo = factory.newClientInfo();
```

Uniform Resource Name (URN)

```

clientInfo.load(request);
%>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="main" title="table">
<p>
<% if clientInfo->getTableSupported() == true; { %>
<table columns="3">
<tr> <td>Rome</td><td>sunny</td><td>28&#176;C</td> </tr>
<tr> <td>Berlin</td><td>cldy</td><td>21&#176;C</td> </tr>
<tr> <td>Oslo</td><td>rain</td><td>15&#176;C</td> </tr>
</table>
<% } else { %>
<%= clientInfo->getDefaultBullet(); %>Rome<br/>
sunny 28&#176;C<br/>
<%= clientInfo->getDefaultBullet(); %>Berlin<br/>
cloudy 21&#176;C<br/>
<%= clientInfo->getDefaultBullet(); %>Oslo<br/>
rain 15&#176;C<br/>
<% } %>
</p>
</card>
</wml>

```

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.558 TelephonyLinksSupported Method

Use

The `getTelephonyLinksSupported` method specifies whether or not you can use a link to directly dial a telephone number. If a mobile device meets this condition and supports telephony links, a WTAI (Wireless Telephony Application Interface) link is used.

If the device does not meet the condition, the phone number should be displayed on the screen, so the user can dial it afterwards.

WAP-enabled Nokia devices allow you to dial phone numbers displayed on the screen by using the [USE NUMBER] function. Since this dial operation does not use WTAI links, the `getTelephonyLinksSupported` method returns the value *false* for these devices.

Example

```

<%@ page import="com.sap.mobile.clientinfo.*" %>
<%! String contentType; %>
<%
    ClientInfo clientInfo = ClientInfoFactory.newInstance().newClientInfo();
    clientInfo.load(request);
    contentType = clientInfo.getContentType();
%>
<%response.setContentType("text/vnd.wap.wml");%>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

```

Uniform Resource Name (URN)

```

<wml>
<card id="karte1" title="Dialling Example">
<% if (clientInfo.isTitleSupported() == false)
{ %>
    <p align="center">
<% if (clientInfo.isEmphasizedSupported() == true) { %>
    <em>Dialling Example</em>
<% } else { %>
    Dialling Example
<% } %>
</p>
<% } %>
<p>
<% if (clientInfo.isTelephonyLinksSupported() == false)
{ %>
Dialling is not supported on this device
<% } else { %>
use softkey for dialling!
<a href="wtai://wp/mc;+496227764900">+49 6227 7 64900</a>
    <do type="options" label="Dial">
        <go href="wtai://wp/mc;+496227764900"/>
    </do>
<% } %>
</p>
</card>
</wml>

```

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.59 isTextStylesSupported Method

Use

The `isTextStylesSupported` method specifies whether or not the browser can format text using tags such as `` or `<small>`.

If these format tags are used, although the browser cannot interpret them, no problems are caused in familiar devices. However, these tags unnecessarily use up valuable memory space.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.60 isTitleSupported Method

Use

The `isTitleSupported` method of the `ClientInfo` interface specifies whether or not the title attribute of the WML tag `<card>` causes the title to appear at the top of the screen.

On some devices, for example Nokia devices, specifying a title in the WML tag `<card>` displays a header in the Web application title bar.

However, an Openwave browser does not support this function on every device. In this case, no header appears in the device's user interface. To make the title appear on such devices, it must be included in the body text of the document.

Example

The following JSP code shows how to use the `isTitleSupported` method:

```
<%@ page import="com.sap.mobile.clientinfo.*" %>
<%@ page import="WeatherData" %>
<%
    int prio;
    ClientInfo clientInfo = ClientInfoFactory.newInstance().newClientInfo();
    clientInfo.load(request);
%>
...
<%response.setContentType("text/vnd.wap.wml");%>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="karte1" title="Table Example">
<% if (clientInfo.isTitleSupported() == false)
{ %>
    <p align="center">
<% if (clientInfo.isEmphasizedSupported() == true) { %>
        <em>Table Example</em>
<% } else { %>
        Table Example
<% } %>
    </p>
<% } %>
...
</card>
</wml>
<% } %>
```

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.61 getTitleWidth Method

Use

If a mobile device does not support proportional fonts and therefore each character has the same width, the value that is returned by the `getTitleWidth` method represents the maximum number of characters a title can be displayed with.

This number might differ from the actual character number for a title when using proportional fonts. In this case, the value that is returned by the `getTitleWidth` method represents the maximum number of characters with an average character width.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.5.62 **getUserAgent Method**

Use

The value returned by the `getUserAgent` method represents the value of the HTTP request header `USER-AGENT`. This header is sent to the browser by a mobile device in every HTTP request and contains information on which browser type and browser version is used by the requesting device.

Example

For example, Ericsson's mobile telephone R380 passes the character string *R380 2.0 WAP1.1* to the Web server. This character string is used to uniquely identify the browser of the mobile device and, by using matching in the device recognition process, ensures that only the appropriate properties for a mobile device are made available.

[ClientInfo Interface \[Page 33\]](#)

3.4.2.12.2.6 **The Administrator's Guide to the Device Recognition Process**

Purpose

The following explanation provides System Administrators with a guide to installing and executing `ClientInfo` for Java Web applications (delivered as `clientinfo.war`). The `ClientInfo` Web application is an extension of the Java-enabled Web server and is implemented in Java in the package `com.sap.mobile.clientinfo`. This Web application provides the device properties of mobile devices like pocket PCs and WAP-enabled mobile telephones at runtime.

Among other tasks, System Administrators can change the Deployment Descriptor `web.xml` file. They can also change values already specified for the properties of mobile devices and add new mobile devices to the device recognition process.

Features

Installing a Web application

- Modifying the Deployment Descriptor

System Administrators can adapt the deployment of the Web application to the Web server structure of their own enterprise by changing the parameters of the Deployment Descriptor element `<servlet>`.

For more information on the Deployment Descriptor, refer to [The Deployment Descriptor \[Page 33\]](#).



When assigning file names, field names, and so on, bear in mind that Java is case-sensitive.

Including other mobile devices in the device recognition process

The innovations in the mobile devices market sector ensure that there are always new devices on the market. This in turn means that not all commercially available mobile devices with their properties and values can be contained in the delivered XML documents.

Uniform Resource Name (URN)

To add additional devices with their special properties, System Administrators can create new XML files.

1. Change the properties predefined by SAP and thus adapt the guidelines for Web applications to the needs of your enterprise.
2. Include new devices by adding more XML files.

For details on maintaining and enhancing these XML documents, see the following:

- [Modifying Device Properties \[Page 33\]](#)
- [Adding New Mobile Devices \[Page 33\]](#)

3.4.2.12.2.6.1 The Deployment Descriptor

Definition

In Java programming, **deployment** refers to the process of installing the Web application in the Web server environment. Among other things, the deployment process ensures that the files and other resources are stored in the right places and that all the information the Web server needs to run the application is available.

The relevant web.xml file contains the "Web Application Deployment Descriptor" of the Java Web application. This descriptor (an XML document) defines the configuration of the Web application and contains all the information the Web server needs. This information is needed to install (deploy) the **clientinfo.war** file (delivered by SAP) to the Web server environment and to control, for example, saving the individual files in the correct directories. This **Web ARchive (.war)** file is a packed Web application.

Delivering the Web application as the Web Archive file clientinfo.war makes deployment easier, since all the files needed for this Web application are installed in the correct hierarchical structure on the Web server.

Use

To deploy a Web application successfully the system requires certain information, which must be formulated in the Deployment Descriptor web.xml file. Before you can set up a servlet you must specify the servlet element of the Deployment Descriptor web.xml more exactly.

The configuration information, which the Deployment Descriptor of the Java Web application ClientInfo describes, includes :

- The definition of the servlet and its initialization parameters
- MIME type mappings (lists of authorized MIME types)

The *ClientInfoInitializer* servlet, however, requires additional information that is passed to it initialization. This information is defined in the Deployment Descriptor element <servlet>.

The initialization parameters of the *ClientInfoInitializer* servlet include:

1. Parameters specifying the class to be implemented using the *ClientInfo* interface. .

```
...
<init-param>
<param-name>com.sap.mobile.clientinfo.ClientInfoImpl</param-name>
<param-value>com.sap.mobile.clientinfo.DefaultClientInfo</param-value>
</init-param>
...
```

Uniform Resource Name (URN)



If you use the SAP implementation, the *DefaultClientInfo* is used by default for implementing the *ClientInfo* interface – that is, the above parameters need not be changed. If, however, you use your own implementation of the *ClientInfo* interface, you must assign the appropriate parameter value to *ClientInfoImpl*.

3. Parameters specifying the root directory of the device recognition data

```
<init-param>
<param-name>com.sap.mobile.clientinfo.Home</param-name>
<param-value>C:\tomcat\webapps\clientinfo\mobile</param-value>
</init-param>
```



Note that you can save the device recognition data in a different directory. It makes sense to do this if you want to add new devices and want to store the new files temporarily in another directory for test purposes.

4. Parameters for specifying the trace level

```
<init-param>
<param-name>com.sap.mobile.clientinfo.TraceLevel</param-name>
<param-value>1</param-value>
</init-param>
```

The trace level describes the information depth of a log file and is a value from 0 (not logged) to 4 (debug). For the *ClientInfo* Web application the trace level is set by default to the value 1 (brief).

It may be appropriate in some circumstances – for example, for technical support – to raise the trace level to 4. This would then increase the amount of information contained in the log file, since only then could the error messages accompanying problems be interpreted. However, to set the trace level to a high value all the time would be bad for the performance of the Web application.

Moreover, if you assign a value of 1 to the sub-element `<load-on-startup>` causes the servlet *ClientInfoInitializer* to be executed first.

You can edit the parameters of the Deployment Descriptor element `<servlet>` and thus adapt the directory and data structure to your own Web server. For example, you can change the root directory for the device recognition data of the Java Web application *ClientInfo* specified by SAP, and set up an appropriate structure for your own enterprise.

For more details on how to do this, refer to [Changing Deployment Descriptor Elements \[Page 33\]](#).

3.4.2.12.2.6.2 Changing Deployment Descriptor Elements

Use

When you deploy the Web application in the structure of the Web server in your own enterprise, you must adapt the [Deployment Descriptor web.xml \[Page 33\]](#) file for this structure to your own needs.



We also advise you to store the .cap files with the device properties temporarily in another directory for test purposes – for example, when adding new devices.

Uniform Resource Name (URN)

To do this, you must edit the XML document `web.xml` containing the parameters of the Deployment Descriptor element `<servlet>`.

Procedure

- Open the XML document `web.xml`, stored in the `WEB-INF` sub-directory of the root directory, in an appropriate Editor – either an XML Editor or Microsoft Notepad.
- Enter your changes – for example, enter a new path for the root directory.
- Save your changes.



After you change the Deployment Descriptor you must execute the Java Servlet Engine again.

Result

You have customized your Web application and tailored the deployment of the Web application to your own enterprise.

3.4.2.12.2.6.3 Modifying Device Properties

Use

To adapt the device properties for a specific mobile device to the layout guidelines of your enterprise, you must edit the associated XML file and then store the changed values in the XML document with the **.cap** extension.

The *.cap XML files for the device types are stored on the Java-enabled Web server in the sub-directory `.../mobile`.

Procedure

1. In the XML Editor, open the XML document with the **.cap** extension containing the device properties of a specific device type. (You can also use the widely available Microsoft Notepad).
2. Change the appropriate data or add new values for a specific device type. (See the example for more details). For a summary of all available device properties, see the template file **template.cap** and the Document Type Definition (DTD) files **devcap.dtd**.
3. Save your entries.



Note that you can also adapt the display of the Web application to the look and feel of your enterprise. To do this, change the values suggested by SAP returned by the methods of the **ClientInfo** interface – for example `getDefaultActionDesign` and `getDefaultBullet` – to the design laid down by your enterprise.

In some cases, you may also want to change the properties preset by the device – for example, if a Web application is displayed in landscape format on a handheld device like a

Uniform Resource Name (URN)

pocket PC. Normally, the `getPixelHeight` and `getPixelWidth` methods return values that display the application in portrait format.



If you have deleted a property, its default value is returned to the Java application at runtime.

In any case, you must restart the Java Servlet Engine after changing the device properties.

Result

You have updated the XML document `.cap`, which provides the device properties for a specific mobile device.

Example

In the following example, the values delivered by SAP – “---“ of the method `getDefaultBlockSeparator` and “>” of the method `getDefaultBullet` – are replaced by the string “***” and the “-“ character (– in the source code) in the `.cap` file.

```
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by SAP AG (SAP AG) -->
<!DOCTYPE device SYSTEM "devcap.dtd">
<device>
<backHardWired>true</backHardWired>
<backLabel>>false</backLabel>
<backToAnyUrlSupported>true</backToAnyUrlSupported>
<cacheEnabledByDefault>true</cacheEnabledByDefault>
<charHeight>4</charHeight>
<charWidth>16</charWidth>
<contentType>wml</contentType>
<contentTypeVersion>1.1</contentTypeVersion>
<deviceCategory>Phone</deviceCategory>
<deviceName>siemensGeneric</deviceName>
<formFactor>phone</formFactor>
.....
<defaultActionDesign>softkey</defaultActionDesign>
<defaultBlockSeparator>***</defaultBlockSeparator>
<defaultBullet>&#150;</defaultBullet>
<defaultFormStyle>onePage</defaultFormStyle>
<defaultMenuStyle>linkList</defaultMenuStyle>
<defaultPageLayoutDesign>sequential</defaultPageLayoutDesign>
</device>
```

3.4.2.12.2.6.4 Adding a New Mobile Device

Use

To make device properties for additional mobile devices available, create a new XML document with the extension `.cap` and save it in the appropriate directory.

The XML file for each device type is stored on the Java-enabled Web server in the sub-directory `.../mobile` of the root directory.



When you add new devices, we recommend that you temporarily copy the files describing the device properties (that is, all the files in the `/mobile` sub-directory) to another directory for test purposes. To do this, you must change the

Deployment Descriptor parameters appropriately. (See [Changing Deployment Descriptor Elements \[Page 33\]](#) for more information).

Procedure

1. To create a new device type, choose the template provided, **template.cap**, stored on the Java-enabled Web server under *.../mobile*. The **template.cap** file lists all the available device properties. They offer an overview of all available device properties, as well as providing help when you want to enter device-specific values for these properties. All available device properties are also stored in the DTD file **devcap.dtd**. You only need to define properties for a specific device.



If you do not define properties the default values are sent to the Java application at runtime.

2. In the XML Editor, open the XML document **template.cap**, which contains the device properties of a specific device type. (You can also use the widely available Microsoft Notepad).
3. Enter the appropriate data – that is, the values you want to assign to the device properties.
4. Save the new file with a meaningful name and the extension **.cap** – for example, **siemensM35.cap**.
5. Then edit the configuration file **devices.xml** saved in the same directory in an XML Editor. You have to add another `<device/>` tag to this file. To do this, enter:
 - a. The ID number
 - b. The string contained in the HTTP header “userAgent”
 - c. The priority (specifies the exactness with which the device can be identified). See [Device Recognition Mechanism \[Page 33\]](#)
 - d. The MIME type passed in the HTTP header “accept” by the requesting device
 - e. The name of the device



```
<device>
<id>Y_SiemensS35</id>
<userAgent>SIE-S35/1.0 UP</userAgent>
<priority>1</priority>
<accept>text/vnd.wap.wml</accept>
<deviceName>SiemensS35</deviceName>
</device>
```



The device name `<deviceName/>` in the **devices.xml** file must match the name of the associated XML file, **.cap**. Choose a meaningful name for the name of the device in the configuration table. The ID number must begin with an upper case Y or Z, followed by an underscore, and then by a meaningful string representing the name of the device. The ID numbers should begin with a Y or Z; otherwise they will be overwritten during updating.



When maintaining contents or names of files, fields and so on, bear in mind that Java is case-sensitive.

In any case, you must restart the Java Servlet Engine after changing the device properties.

Result

You have added a specific device to the device recognition mechanism.

3.4.3 User Management Engine

Purpose

SAP Enterprise Portal 6.0 uses the User Management Engine (UME) 4.0 to enable integration of the portal with a existing LDAP or user management solution. The UME is used by the *SAP NetWeaver* platform.

This document describes how to develop customized applications with the UME that is provided by the [SAP Web AS Java \[Page 33\]](#) and the [SAP Enterprise Portal \[Page 33\]](#).

From the application point-of-view, the UME provides three main functions:

- **Authentication:** Which user is logged in
- **Authorization:** What the user is allowed to do
- **Profile:** Details of user like user name, address and so on

See the *SAP Web AS Java* documentation [Identity Management \[External\]](#) for more details about the authentication concept. This document is also available on the *SAP Help Portal* at <http://help.sap.com>.

3.4.3.1 SAP Web AS Java

See the [SAP Web AS Java UME \[External\]](#) documentation for details about installation and administration. This document is also available on the *SAP Help Portal* at <http://help.sap.com>.

User Authentication and Single Sign-On

There are several mechanisms for user authentication available on the *SAP NetWeaver* platform. If you have several systems in your system landscape, the Single Sign-On (SSO) environment is a useful feature, to reduce the number of logon procedures a user usually has to perform.

The *SAP Web AS Java* is the underlying technology for authenticating users with *SAP NetWeaver*. The following table gives an overview of the available user authentication services and if the service is used directly for user authentication or for SSO.

Mechanism	User Authenti-	Single Sign-	ABAP	Java
-----------	-------------------	-----------------	------	------

Uniform Resource Name (URN)

	ation	On		
User ID and password	x	x	x	x
Secure Network Communications (SNC)	x	x	x	
SAP logon tickets		x	x	x
SSL and X.509 client certificates	x	x	x	x
Pluggable Authentication Services (PAS)	x		x	
Security Assertion Markup Language (SAML)		x		x
Java Authentication and Authorization Service (JAAS)	x	x		x

For more details on SSO refer to the [Single Sign-On in a Complex System Landscape \[External\]](#) or refer to the *SAP Help Portal* at <http://help.sap.com>.

3.4.3.1.1 Authentication

The logon stacks enable you to choose different combinations of authentication types for every application you create, and for each of the components on the server with applied security restrictions.

Interface IAuthentication

The *SAP WebAS Java* provides an API to check if a user is logged in, to enforce that a user is logged in and to get the logged in user object.

The interface has following methods:

```
public interface IAuthentication {

    //Returns the logged on user or null, if no user is logged on.
    public IUser getLoggedInUser(HttpServletRequest req,
                                HttpServletResponse resp);

    /*
     * Checks if a user is logged on and returns the user id if it is.
     * If the user is not logged on, a logon page is displayed,
     * written as ServletResponse.
     */
    public IUser forceLoggedInUser(
        HttpServletRequest req,
        HttpServletResponse resp)
        throws UserManagementException;

    //Logs the user out
    public void logout(
        HttpServletRequest req,
        HttpServletResponse resp);
}
```

Example: Enforcing Logon

```
IUser user =
    UMFactory.getAuthenticator().forceLoggedInUser(request, response);
if (user == null)
```



```
return;
```

The `user` object can be used for access control and to get the profile of the user.



The method `forceLoggedInUser()` changes the response if the user is not logged on.

To avoid exceptions an application must stick to the following recommendations:

- do not write to the response before calling method `forceLoggedInUser()`.
- do not write to the response after calling method `forceLoggedInUser()` when the method returned value `null`.

Session Handling for SAP WebAS Java Applications

If a *SAP WebAS Java* application stores confidential or user relevant data in the *session* context, the application has to make sure that the data/session is destroyed when the user logs off. The application has to check if a user is logged on at every request with the method `getLoggedInUser()`.

Whenever a user logs on, a reference to the logged on user is stored in the *session* context. If the session already contains a reference to another user (this includes the case that the session contains a user reference but the user is not logged on), use the method `forceLoggedInUser()` to initiate a new log on and to cancel the existing session. When the user logs off, the SSO cookie is removed and the session is closed.

For a good performance of the method `getLoggedInUser()`, the UME caches the information to verify the log on status of user.



See the *Sun Microsystems* servlet specification for more details about the HTTP session object when you have to create a *SAP WebAS Java* application.

In the *SAP Enterprise Portal* the *session* is controlled by the portal. So the portal application does not have to take care about user data stored in the *session* context itself.

Single Sign-On (SSO)

Authentication with SSO works as follows:

- After a user is logged on, an encrypted cookie is created for the user.
- In the following requests, this cookie can be used for SSO. The method `getUser()` verifies the cookie and retrieves the available user information.

The method `forceLoggedInUser()` works in the same way.

- When the user is not logged on or the cookie is for any reason invalid and method `forceLoggedInUser()` is called, the method will automatically display the log on page. The requested URL (for example, servlet, HTML page, JSP and so on) is passed on to the log on page. When the user is logged on again, the requested URL is called.

3.4.3.2 SAP Enterprise Portal

See the *SAP Enterprise Portal UME* documentation for details about configuration and administration. The document *Authentication on the Portal* contains details about Single Sign-on (SSO).

Both documents are also available on the *SAP Help Portal* at <http://help.sap.com> → *SAP NetWeaver* → *People Integration* → *Portal* → *Portal Architecture* → *Security and User Management* → *User Management Engine (UME)*.

3.4.3.2.1 LoginModule Example

Purpose

The *SAP Enterprise Portal 6.0* provides a flexible mechanism to plug-in custom authentication mechanisms and authentication configuration for a portal content.

This section explains the basic concepts where the customized authentication mechanisms can be plugged-in, how the flexible architecture can be used and how it is assembled.

Prerequisites

You should be familiar with the following services and tools:

- Java Authentication and Authorization Service (JAAS).
Documentation for the JAAS service can be found on the Sun Microsystems Inc. Java developer network site.
- Experiences with the Java programming language.
- Basic concepts of the *SAP Enterprise Portal*, iViews and authentication schemes.
Documentation on these issues can be found on the *SAP Help portal*.

Structure of this Section

Section [Custom Password Based Authentication \[Page 33\]](#) covers the easiest way to customize the logon. The `LoginModule` interface has to be implemented and a few configuration changes have to be done. The portal will use the standard logon screen.

Section [Changing the Login Screen \[Page 33\]](#) describes how to replace the standard portal logon screen by a customized logon screen.

Section [Advanced Authentication Example \[Page 33\]](#) describes the advanced features of the authentication architecture like, how to evaluate http information and how to deal with cookies.

3.4.3.2.1.1 Customized Password Authentication

The authentication by password is the most common authentication method. The *SAP Enterprise Portal* provides several password based authentication features, like for the following services:

Uniform Resource Name (URN)

- All supported LDAP directories.
- The User Management Engine (UME) 4.0 proprietary database repository.
- The SAP ABAP engine.

Customized authentication is necessary in following use cases:

- Proprietary user repository (for example, LDAP compliant)
User records can be synchronized with the portal user repository but passwords cannot. This is the case, if the one-way hash algorithm used to store passwords is not identical for both user repositories. The stored hash values cannot be copied from one repository into the other and the password cannot be restored from the hash value. The *SAP Enterprise Portal* user administration can use the synchronized user record in the portal user repository but the authentication must be performed with the original repository.
- Hardware device generates a PIN for authentication
Usually these hardware devices are synchronized with a central server which verifies the pins. In this case the customized authentication would establish a connection with the central server and verify the pin.

In this section we describe the basic concept of customized authentication with the `LoginModule` interface and which steps are necessary to integrate it into *SAP Enterprise Portal*.

3.4.3.2.1.1 Customized Authentication Implementation

In this section a customized authentication is implemented that will accept a password if it is equal to the user name after a *cyclic right-shift* by one character. The comparison is not case sensitive.

The class of the example is called `com.sap.security.demo.DemoLoginModule`.

Implementing the LoginModule Interface

The `LoginModule` interface provides the method necessary to implement the customized authentication. In the following we describe the methods that have to be implemented.

checkPasswd(String user, char [] pwd) Method

This method verifies the password.



The password parameter `pwd` comes in a *character* array in compliance with the JAAS standard.

The method is implemented as follows:

```
/**
 * This function verified if the password is a cyclic
 * right shift by one character of the lowercase user
 * name
 */
protected boolean checkPasswd (String name, char [] pwd)
{
    if (name==null || pwd==null || name.length()==0 || pwd.length()==0)
    {
```

Uniform Resource Name (URN)

```

        return false;
    }
    name = name.toLowerCase();

    // cut off the first character
    String strPwd = new String (pwd, 1, pwd.length-1);

    // compare if the first character of the password
    // is the last of the user name
    if (pwd[0]!=name.charAt(name.length()-1))
        return false;

    return name.substring (0,name.length()-1).equals(strPwd);
}

```

initialize() Method

This method is called when the service is initialized, for example when the portal is started.

```

Subject      _subject      = null;
CallbackHandler _ch        = null;

public void initialize (Subject subj, CallbackHandler ch,
                        Map sharedState, Map options)
{
    _subject = subj;
    _ch      = ch;
}

```



To save resources the portal cannot keep the login context. Therefore the settings are not available in the `sharedState` map for an entire *logon/logoff* cycle. A new `sharedState` map is instantiated before the methods `login()` or `logoff()` are called.

login() Method

This method is called, when the user chooses the *Logon* button on the logon screen. The `login()` method gets the username and password. It checks the password and returns `true`, when the password is correct, or an exception when the password is incorrect.

```

public boolean login() throws LoginException
{
    Exception      exception_on_the_way      = null;
    PasswordCallback pc = new PasswordCallback ("Password:", false);
    NameCallback    nc = new NameCallback ("User:");
    Callback []     mycallbacks = new Callback [] { nc, pc };

    try {
        _ch.handle (mycallbacks);
    }
    catch (IOException e) {
        exception_on_the_way = e;
    }
    catch (UnsupportedCallbackException e) {

```

Uniform Resource Name (URN)

```

        exception_on_the_way = e;
    }

    String name = nc.getName();
    char [] pwd = pc.getPassword();

    if (name.length()==0)
        throw new LoginException (MISSING_UID);

    if (pwd.length==0)
        throw new LoginException (MISSING_PASSWORD);

    if (exception_on_the_way!=null) {
        // A productive application should write an entry
        // into the trace here
        exception_on_the_way.printStackTrace ();
        throw new LoginException ("Could not handle callbacks");
    }

    if (!checkPasswd (name, pwd)) {
        throw new LoginException (USER_AUTH_FAILED);
    }
    else {
        _bSucceeded = true;
        _auth_user = name;
    }

    return true;
}

```

If a `LoginException` exception is thrown in the method `login()`, like shown in the example above, the standard logon page of the portal displays the standard error messages for the standard error cases, like *user or password incorrect*.

To display a specific message on the logon page, a

`javax.security.auth.login.LoginException` with a defined error constant has to be thrown. The error constants are defined as *String* constants in the core class `com.sap.security.core.logon.imp.SecurityPolicy`. Since the core class is not part of the published UME API the error codes have to be defined as follows:

```

public final static String MISSING_UID      = "MISSING_UID";
public final static String MISSING_PASSWORD = "MISSING_PASSWORD";
public final static String USER_AUTH_FAILED = "USER_AUTH_FAILED";
public final static String USERID_NOT_FOUND = "USERID_NOT_FOUND";
public final static String ACCOUNT_LOCKED_ADMIN = "ACCOUNT_LOCKED_ADMIN";
public final static String ACCOUNT_LOCKED_LOGON = "ACCOUNT_LOCKED_LOGON";

```

commit() Method

This method is called, when the `login()` method returned `true`. It provides the identity of the authenticated user.

```

public boolean commit ()
{
    if (_bSucceeded == false) {
        return false;
    }
}

```

Uniform Resource Name (URN)

```
else {
    // add a Principal (authenticated identity) to the Subject
    final String final_name = _auth_user;
    _subject.getPrincipals().add (new Principal () {
        public String getName ()
        {
            return final_name;
        }
    });
    return true;
}
```

The user name used to create the `Principal` object has to be the logon user ID of an existing portal user. The SAP Enterprise Portal will instantiate a `com.sap.security.api.IUser` object with the method `IUserFactory.getUserByLogonID(String logonuid)`.

Deploying the Customized Authentication

Build a Java Archive (JAR) file that contains the customized authentication implementation and other classes that the implementation needs.

For this example the class files of the `com.sap.security.demo.DemoLoginModule` implementation is put into the `demolm.jar` file. The `demolm.jar` file has to be copied into the folder `<j2ee home>/cluster/server/additional-lib`.

To register the customized authentication the configuration of the portal has to be changed. See section [Configure the Portal for Customized Authentication \[Page 33\]](#) for more details.

3.4.3.2.1.1.2 Configure the Portal for Customized Authentication

When the JAR file with the classes of the customized authentication implementation (`demolm.jar` in this example) has been copied to the folder `<j2ee home>/cluster/server/additional-lib` the JAR file has to be added to the following configuration files.

Changes in file `library.txt`

The classes in the `demolm.jar` file have to be *visible* to the UME classes and vice versa. This is done by adding the JAR file to the definition of the UME libraries in the file `<j2ee home>/cluster/server/managers/library.txt`.

Append the name of the JAR file as follows (the added characters are written in bold):

```
library com.sap.security.ume
com/sap/security/api/com.sap.security.api.jar;
com/sap/security/api/com.sap.security.api.perm.jar;
com/sap/ip/basecomps/BaseComps.jar;
com/sap/security/api/com.sap.security.core.jar;
com/sap/security/api/com.sap.security.core.tpd.jar;
com/sap/security/demolm.jar
```

Uniform Resource Name (URN)

Changes in file *authschemes.xml*

The customized authentication implementation has to be added as new *authscheme* section to the *authscheme* structure defined in the file `authschemes.xml`.

New *authscheme* section:

```
<authscheme name="myNewLogon">
  <loginmodule>
    <loginModuleName>
      com.sap.security.demo.DemoLoginModule
    </loginModuleName>
    <controlFlag>REQUISITE</controlFlag>
    <options></options>
  </loginmodule>
  <priority>20</priority>
  <frontendtype>2</frontendtype>
  <frontendtarget>com.sap.portal.runtime.logon.certlogon</frontendtarget>
</authscheme>
```

The standard *authscheme* entry of the portal is called *uidpwwlogon*. The new *authscheme* entry is identical to *uidpwwlogon*, except for the *authscheme* name and *loginModuleName*.

Using the Customized Authentication as Default Authentication

To use a customized *authscheme* as standard portal authentication, the *authscheme* parameter in the *authscheme-refs* section has to be changed to the name of the customized *authscheme*.

To use the *myNewLogon* *authscheme* the *authscheme-refs* entry has to be changed as follows:

```
<authscheme-refs>
  <authscheme-ref name="default">
    <authscheme>myNewLogon</authscheme>
  </authscheme-ref>
</authscheme-refs>
```



Refer to **SAP Note 686538** for details about saving the settings in the `authschemes.xml` file before applying a Support Package to the SAP Enterprise Portal.

Using the Customized Authentication for Specific iViews

With the portal user interface in section *Portal Content*, you can assign the customized authentication to certain iViews.

The customized authentication can also be specified in the deployment descriptor `portalapp.xml` of an iView.

Running the Customized Authentication

After all the changes on the portal configuration are finished, restart the server and logon to the portal.

For more details about creating an iView and JSP refer to the PDK.

The customized login screen has to follow certain rules in naming the form and input fields.

- The input field for the user name must have the name `j_user`.
- The input field for the password must have the name `j_password`.
- The hidden input field `login_submit` has to be set to true (=1) to tell the portal that a logon takes place.
- To use another *authscheme* than default, the customized *authscheme*, that refers to the name in the `authschemes.xml` file, has to be provided in the hidden input field `j_authscheme`.

[illegible]


```
</table>
</form>
```

3.4.3.2.1.3 Advanced Authentication Example

The portal allows the access to the http request for all custom logon modules during the authentication process. With access to the http request it is possible to read and set cookies or http header values, for example to reject certain namespaces.

In the following chapter we extend the example with an access to the http request.

3.5 Changes in the LoginModule Implementation

We extend the `LoginModule` implementation, described in section [Customized Authentication Implementation \[Page 33\]](#), that in addition to the password the IP address of the client has to be in a predefined range.

initialize() Method

For the further coding, we need the *options* parameter, which is available in `initialize()`. We store the *options* parameter locally.

Extended `initialize()` method:

```
Subject      _subject      = null;
CallbackHandler _ch        = null;
Map _options  = null;

public void initialize(Subject subj, CallbackHandler ch,
                      Map sharedState, Map options)
{
    _subject = subj;
    _ch      = ch;
    _options = options;
}
```

login() Method

We extend the existing `login()` method, because, we still want to check the password. Therefore we create the `com.sap.security.demo.DemoLoginModule2`, inherit a new `LoginModule` class from our existing `LoginModule` class from section [Customized Authentication Implementation \[Page 33\]](#) and overwrite the method `login()` method the following way:

```
Exception on_the_way = null;
String client_ip_str = null;
String ip_range_str = (String)_options.get("com.sap.security.demo.ip_ma
sk");
String ip_match      = (String)_options.get("com.sap.security.demo.ip_ma
tch");

// First we check if the password login
// is successful
boolean rc = super.login ();
```

Changes in the LoginModule Implementation

```

// In addition we perform the check whether
// The ip address is within a given range

// Get ip-range from options
byte [] iprange_as_byte_array = getIPAsByteArray (ip_mask);
// Get ip-address of client
byte [] client_ip = null;
// Get ip-match
int [] ipmatch = getIPAsIntArray (ip_match);

try {
    client_ip_str = getClientIP ();
} catch (UnsupportedCallbackException e) {
    on_the_way = e;
} catch (IOException e) {
    on_the_way = e;
}

client_ip = getIPAsByteArray (client_ip_str);

if (on_the_way!=null) {
    on_the_way.printStackTrace ();
    throw new LoginException ("Exception occured");
}

if (!ip_address_in_range (client_ip, iprange_as_byte_array, ipmatch))
    throw new LoginException ("IP-address " + client_ip_str +
        " is not in range" + ip_range_str);

return rc;

```

Utility Methods

The method `ip_address_in_range()` checks if the IP address of the client is in a given range. Therefore, it first flips all bits in the array `iprange_as_byte_array` and performs then a logical AND with the IP address. The result of the AND operation is zero, when all bits are in the allowed range.

The method `getClientIP()` gets the client IP. To get access to the http request object, we call the `handle()` method with an instance of `WebCallback`. The `handle()` method is the interface of the *LoginModule* to the calling environment and enables the access to the object data. Refer to the JAAS documentation for more details.

```

private boolean ip_address_ok (byte [] client_ip,
                                byte [] iprange_as_byte_array,
                                byte [] ipmatch)
{
    for (int ii=0; ii<4; ii++) {
        if (ipmatch[ii]!=(client_ip[ii] & iprange_as_byte_array[ii]))
            return false;
    }
    return true;
}

private String getClientIP ()

```

Changes in the LoginModule Implementation

```
throws UnsupportedOperationException, IOException
{
    WebCallback wcb = new WebCallback ();
    _ch.handle (new Callback [] { wcb });

    HttpServletRequest req = wcb.getRequest();

    return req.getRemoteAddr ();
}
```

Configuration

To use the `com.sap.security.demo.DemoLoginModule2` implementation we have to add an *authscheme* to the `authschemes.xml` file.

The entry has following format:

```
<authscheme name="myNewLogon2">
  <loginmodule>
    <loginModuleName>
      com.sap.security.demo.DemoLoginModule2
    </loginModuleName>
    <controlFlag>REQUISITE</controlFlag>
    <options></options>
  </loginmodule>
  <priority>25</priority>
  <frontendtype>2</frontendtype>
  <frontendtarget>com.sap.portal.runtime.logon.certlogon</frontendtarget>
</authscheme>
```

The priority of this authentication is higher, because the authentication mechanism is stronger.

3.5.1.1 User

The *SAP WebAS Java* and the *SAP Enterprise Portal* provide interfaces to get details about the current user.

User Management Factory

The user management factory gives access to the user management functions. The user management factory is in the following package:

```
com.sap.security.api.UMFactory
```

Interface IUser

The `IUser` interface provides read access to the available user information.

Examples for user information:

- Company the user belongs to.
- Profile information, like full name and address)
- Authorization information. Which rights does the user have.

Changes in the LoginModule Implementation

The `IUserMaint` interface extends the `IUser` interface and provides methods to change the user information. One reason for the separation is the performance. Most applications only need read access. The `IUser` interface does not have to deal with the overhead necessary to change a user and is therefore faster.

Getting a User Object

The user management factory provides several methods to get the `IUser` object, for example, get user by user name. For all available methods, please refer to the Javadoc.

Example:

```
//Get an 'IUserFactory'-object to instantiate user-objects.
IUserFactory userFactory= UMFactory.getUserFactory();
IUser myUser = userFactory.getUser(request.getUser().getUserId()
);
```

Accessing the Logon ID of a User

In EP 5.0 it was possible to get the logon ID with the `getUid()` method of the user factory. This method is deprecated in EP 6.0 due to the fact that a user can be associated with several logon accounts. The method `getUniqueId()` is a substitution for `getUid()`, but the returned value has to be parsed to get the logon ID. You can access the logon ID by iterating through the `IUserAccounts` array that is returned by method `getUserAccounts()` T, with method `getLogonUid()`.

```
String logonID = user.getUid();
IUserAccount accounts[] = null;
try {
    accounts = user.getUserAccounts();
} catch (UMException e) {
    response
        .write(
            "<br>Error getting accounts: "
            + e.getLocalizedMessage());
}
if (accounts != null) {
    response.write(
        "<br>Number of Login Accounts: " + accounts.length);
    for (int i = 0; i < accounts.length; i++) {
        response.write(
            "<br>** Login ID #"
            + i
            + ": LogonUID="
            + accounts[i].getLogonUid()
            + ", AssignedUID="
            + accounts[i].getAssignedUserID());
        response.write(
            "<br>Last Login: "
            + accounts[i]
            .getLastSuccessfulLogonDate()
            .toString());
        response.write(
            "<br># Logins: "
            + accounts[i].getSuccessfulLogonCounts());
    }
}
```

3.5.1.1.1 Service User

Service user do not log on interactively. A service user is used, for example, to connect to a remote system with certain rights. Although a service user does not log on interactively, it is authenticated and the attributes contain a valid ticket. [User mapping \[Page 33\]](#) can be defined for a service users as well as assigning a role and general attributes to a service user.

Service users are defined as regular users with their own namespace.

3.6 ServiceUserFactory

The ServiceUserFactory can be accessed as follows:

```
UMFactory().getServiceUserFactory()
```

The ServiceUserFactory provides the method `getServiceUser(String uniqueName)`. The ServiceUserFactory verifies that the specifies user is a service user and if so, returns an IUser object with a ticket attached to a transient attribute.

For a service user only the user profile is stored, not the user itself. This has following advantages:

- No interactive log on possible
authentication will first check in the store of accounts and will not find a user account for a service user no special coding necessary for authentication process
- Initial service users can be provided when the UME tables are created. After creating the UME tables, following users, in this case Knowledge Management service users, are automatically added:

```
"index_service" ,
"subscription_service",
"ice_service",
"collaboration_service",
"timebasedpublish_service",
"notificator_service",
"cmadmin_service",
"action_inbox_service"
```

Security of Service Users

Service users have the rights that are needed to perform a certain task, for example, all necessary permissions to perform all actions, like delete and modify, on an [Access Control List \[Page 33\]](#) (ACL).

Permission check example:

```
IUser createServiceUser(String uid) {
    SecurityManager secman= UMFactory.getSecurityManager();
    if (secman != null) {
        ProtectedCallPermission p=
            new ProtectedCallPermission(createServiceUser, uid);
        secman.checkPermission();
    }
}
```

ServiceUserFactory

```
}

```

The call name acts as target and the user name as action. With this granularity, you can specify exactly which application can instantiate which service users.

Example for a permissions:

```
codeBase ${portal.home}\WEB-INF\portal\&#8230;\private\lib\km.jar {
  grant ProtectedCallPermission createServiceUser
  IndexService,QueueService;
}
```

3.6.1.1 Role

EP 5.0 has a basic role structure, with a unique name for every role. In EP 6.0, roles can be nested in folders and have a unique identifier. Most of the role API methods require the unique role ID as a handle or key to obtain the role information.

The `getRoles()` method of the `IUser` object gets a list of roles as *iterator*. The *iterator* contains the unique IDs of the roles associated with the user. In order to get the actual information about a specific role you have to use the role factory to get an instance of that role. The unique ID is passed to the `getRole()` method of the role factory.

Example:

```
response.write("<br>**** ROLE INFORMATION:");
if (rit.hasNext()) {
    IRoleFactory rfact = UMFactory.getRoleFactory();
    while (rit.hasNext()) {
        String roleName = (String) rit.next();
        IRole role = null;
        try {
            role = rfact.getRole(roleName);
            response.write("<br>Role:" + roleName
                + "<br>Display Name:" + role.getDisplayName()
                + "<br>ID: " + role.getUniqueID()
                + "<br>Uniquename: " + role.getUniqueName()
                + "<br>Description: " + role.getDescription());
        } catch (UMException e) {
            response.write("error: " + e.getLocalizedMessage());
        }
    }
}
```

3.6.1.2 Group

The group API is very similar to the role API for accessing, searching and determining memberships of groups.

3.6.1.3 Searching for Users, Roles and Groups

The UME provides several methods to search for users, groups and roles with different attributes. The search is performed with search filters.

Search for Users

Example to get a user search filter and start search:

```
IUserFactory ufact = UMFactory.getUserFactory();
```

ServiceUserFactory

```
IUserSearchFilter isf = ufact.getUserSearchFilter();  
// Provide the search attributes  
isf.setDisplayName (...);  
// Start search  
ISearchResult sr = fact.searchUsers(isf);
```

Search for Roles

To search for roles is similar to search for users.

Example:

```
IRoleFactory rfact = UMFactory.getRoleFactory();  
IRoleSearchFilter isf = rfact.getRoleSearchFilter();  
// Provide the search attributes  
isf.setDisplayName (...);  
// Start search  
ISearchResult sr = fact.searchUsers(isf);
```

Defining a Search Attribute

The search attribute can be set with `setDisplayName()` method. This method allows exact searches or wildcard searches.

Example:

```
// Provide the search attributes, with wildcards before and after  
// the search string.  
isf.setDisplayName ("*" + mySearchStr + "*",  
                    ISearchAttribute.LIKE_OPERATOR, false);  
  
// Provide the search attributes for an exact search  
isf.setDisplayName (mySearchStr,  
                    ISearchAttribute.EQUALS_OPERATOR, false);
```

Search Result

Depending on the search attribute, the search method returns one or several users or roles. The returned value is a `ISearchResult` object that can be iterated.

To which Role or Group does a User Belong

With the methods `isMemberOfRole()` or `isMemberOfGroup()` it can be determined if a user is a member of a particular role or group. The methods require the unique ID of the role or group.

Example:

```
IRole superRole= null;  
try {  
    superRole= getSingleRole("super_admin", false);  
} catch (UMException e) {  
    response.write(  
        "Error getting role: " + e.getLocalizedMessage());  
}
```

ServiceUserFactory

```

if (user.isMemberOfRole(superRole.getUniqueID(), true))
    response.write(" IS a super admin");
else
    response.write(" IS NOT a super admin");

```

3.6.1.4 User Mapping

User mapping is a feature that has been introduced to SAP products with EP 5.0. This feature provides log on information for users on third party systems. The log on information can simply be user ID and password and additionally information like NT-Domain and language.

In EP 5.0, user mapping is available as a personalization feature for every end user and as an administration feature that allows administrators to map users, groups or roles to specific accounts. In a productive environment, the user mapping feature contains a user interface to administer the data. In the *SAP Enterprise Portal*, the user interface calls the system landscape service to get a list of all systems that apply for user mapping. The system landscape has an user mapping attribute that has to be set accordingly.

The UME for EP 6.0 offers two interfaces to access the user mapping data:

- IUserMappingService
- IUserMappingData

Example:

```

import java.util.HashMap;
import java.util.Map;
import com.sap.security.api.IUser;
import com.sap.security.api.umap.IUserMappingData;
import com.sap.security.api.umap.NoLogonDataAvailableException;
import com.sapportals.portal.prt.component.AbstractPortalComponent;
import com.sapportals.portal.prt.component.IPortalComponentRequest;
import com.sapportals.portal.prt.component.IPortalComponentResponse;
import com.sapportals.portal.prt.runtime.PortalRuntime;
import com.sapportals.portal.prt.service.usermapping.IUserMappingService;

public class UserMapping extends AbstractPortalComponent
{
    public void doContent(IPortalComponentRequest request,
        IPortalComponentResponse response)
    {
        // obtain system
        String systemalias = "system";

        // get user from request
        IUser iuser = request.getUser ();

        // get usermapping service
        IUserMappingService iums = (IUserMappingService)
            PortalRuntime.getRuntimeResources().getService(IUserMappingService
                .KEY);
        IUserMappingData iumd = iums.getMappingData (systemalias, iuser);
        Map map = new HashMap ();
        try {
            iumd.enrich (map);

```


ServiceUserFactory

```

    }
    catch (NoLogonDataAvailableException nldae)    {
    // Error handling
    }

    // In m is all data
    String userid = (String)map.get( "user" );
    String pwd = (String)map.get ( "mappedpassword");
    response.write("hallo" + userid);
    response.write(" " + pwd);
  }
}

```

3.6.1.5 Access Control List (ACL)

The security concept implemented in EP 6.0 allows an administrator (object creator) to create a new ACL object and assign it to the owner.

The ACL feature provides an interface for following functions:

- Create, modify or delete supported permissions for the ACL.
- Create, modify or delete an ACL object for a portal object.
- Add or remove ACL owners.
- Create, modify or delete the permissions for a principal (ACE).
- Check if a user has permission to execute an action.

Application Specific ACL Manager

With the default ACL manager all applications that use ACL have the same namespace. To avoid conflicts with object ID and permission names among applications, the user management factory provides the method `getAclManager(String applicationID)` that returns an application specific ACL manager.

API

The application that uses the ACL API works with following interfaces:

- **com.sap.security.api.acl.IAclManager**

This interface defines all the methods that are required for the general administration of ACL. It allows application to:

- Create, modify, read and delete an ACL object for a portal object.
- Check the permission for a principal on an object.
- Add, remove and get available supported permissions.
- Delete the whole corresponding data for a principal.

- **com.sap.security.api.acl.IAcl**

This interface allows application to:

- Add or remove an ACL owner.
- Check if an user is an ACL owner.
- Create, delete or get ACE.
- Check the user permissions.
- Check the object ID

ServiceUserFactory

- **com.sap.security.api.acl.IAclEntry**

The ACE object contains information about a principal and its permissions. It allows the application to:

- Get the permission and the principal of ACE.
- Check if the ACE has permissions.
- Check if the principal has the required permission.

- **com.sap.security.api.acl.PermissionStatus**

The permission status object returns the status for a given principal if the permission is allowed, denied or undefined. The application gets the ACL Manager from the Portal Runtime (PRT).

Example:

```
IAclService service = (IAclService)
PortalRuntime.getRuntimeResources()
    .getService(IAclService.SERVICE_ID);

IAclManager manager = service.getAclManager();
```

In the UME, the application gets the ACL Manager from the user management factory.

Example:

```
IAclManager aclManager = UMFactory.getAclManager();
```

- **com.sap.security.api.acl.IAclHierarchy**

This interface provides the application an access point to:

- Check the permission for a principal on a list of object IDs which represent the parent objects of the former object.
- Distribute an ACE to the members of a object ID tree. If an ACE for a root object gets changed, this new ACE will be distributed to all members of the sub node of this root. All entries are inherited.

ACL Manager Interface

The ACL Manager administers the ACLs. The ACL manager interface defines methods to administer ACL's and check if a principal has access to an object with a certain permission.

Permissions

A permission is defined by an object type and a permission name separated by a dot (.), for example, `default_type.read`.



A dot is not allowed in the object type, but in the permission name.

A global permission is defined without an object type.

Permissions must be unique within the namespace of the ACL manager. Therefore, for an application specific ACL manager, the permissions have to be unique within the application and for the default ACL Manager, the permission has to be globally unique.

ServiceUserFactory

Object ID

Object IDs must be unique within the namespace of the ACL manager. Therefore, for an application specific ACL manager, the object IDs have to be unique within the application and for the default ACL Manager, the object IDs has to be globally unique.

Example:

```
//Get default ACL Manager
IAclManager manager = UMFactory.getAclManager();

// Get specific ACL Manager
IAclManager manager = UMFactory.getAclManager("Workflow");

//Create some Permissions
manager.addPermission("WorkflowPermission.read", null);
manager.addPermission("WorkflowPermission.write", null);

//Create a Permission Container
List members = new ArrayList(2);
members.add("WorkflowPermission.read");
members.add("WorkflowPermission.write");
manager.addPermission("WorkflowPermission.full_control", members);

//Create an ACL on an objectID
IUser userA;
IAcl acl = manager.createAcl(userA, "WorkflowItemABC");

//Get this ACL again
IAcl acs = manager.getAcls("WorkflowItemABC");

//Delete an ACL
manager.removeAcl(userA, "WorkflowItemABC");

//Delete all info's about a principal (concerning ACL info)
manager.deletePrincipal(userA);

//Create an ACE (Access Control Entry) for user B (user A is ACL Owner)
IAclEntry aclEntry = acl.createAclEntry
    (userA, userB, "WorkflowPermission.read", false);

//Get all ACE's for a special principal
acl.getAclEntries(userB);

//Get all ACE's
acl.getAclEntries();

//check a permission on IAclManager
manager.isAllowed("WorkflowItemABC", userA, "WorkflowPermission.read");

//check a permission on IAcl
acl.isAllowed(userA, "WorkflowPermission.read");

//check a permission on IAclEntry
acl.isAllowed("WorkflowPermission.read");

//Delete an ACL Entry
```

ServiceUserFactory

```
acl.removeAclEntry(usersA, aclEntries);  
  
//Reset the hole ACL (only deletion of ACE's)  
acl.resetAcl(usersA);
```

3.6.2 User Agent Service

Purpose

A *user agent* is an application which is used to browse through the World Wide Web. Web *user agents* can be web browsers and search engine spiders as well as accessibility products like screen readers and braille browsers.

The user agent provides information about itself, when a web site is accessed. The information contains the brand and version of the browser and the operating system the browser is running on.

Example

User agent

Internet Explorer 5.5 running on MS Windows 2000

Provided information

Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0).

How to get the User Agent

Necessary SharingReference Entry in Deployment Descriptor

To use the *user agent service* you have to add a sharing reference entry to the deployment descriptor (file `portalapp.xml`) in the section `<application-config>`.

The entry has following format:

```
<application-config>  
  <property name="SharingReference"  
    value="com.sap.portal.useragent"/>  
  .  
</application-config>
```



When you use another service or services in the portal application, you separate the services by a comma.

Example

```
value="htmlb,com.sap.portal.useragent"
```

Modifying the Desktop and Navigation

Get the Useragent Service

The entry point of the *user agent service* is the interface `IUserAgentService`. You get this interface from the Portal Runtime (PRT) with following command:

```
IUserAgentService userAgentService =
    (IUserAgentService) PortalRuntime
        .getRuntimeResources()
        .getService(
            "com.sap.portal.useragent.useragent");
```

Get the User Agent

To get the *user agent*, you have to get the *user agent name* first. With the *user agent name* and the *user agent service* you get the user agent from the `HttpServletRequest` object. You can use the following commands:

```
HttpServletRequest sRequest = request.getServletRequest();
String userAgentName = sRequest.getHeader("User-Agent");
// Using this information you can use the method getUserAgent()
of
// IUserAgentService to get the user agent object of type IUser
Agent:
IUserAgent userAgent =
    userAgentService.getUserAgent(sRequest.getHeader(userAgentNam
e));
```

For more details about the usage of the `userAgent` methods refer to the PDK.

Get the User Agent Family

The *user agent service* also provides information about the *user agent family* with interface `IUserAgentService`. This interface provides the method `getUserAgentFamily()` that delivers the information. You get this interface with the following commands:

```
IUserAgentFamily userAgentFamily =
    userAgentService.getUserAgentFamily();
/* From this object you get an enumeration of all user agent sets of
type IUserAgentSet that matches the IUserAgent: object: */
Enumeration enum = userAgentFamily.findUserAgentSets(userAgent);
```

3.7 Modifying the Desktop and Navigation

This section describes how to create navigation links in the portal, and includes the following:

- [Navigating in the Portal \[Page 33\]](#)
- [Creating Custom Layouts \[Page 33\]](#)
- [Object-Based Navigation \[Page 33\]](#)

3.7.1 Navigating in the Portal

The portal's navigation service and related interfaces enable you to control how users navigate in the portal.

Modifying the Desktop and Navigation

Essentially, the navigation service creates a tree of navigation nodes for each user who enters the portal. Each node represents specific content, or a collection of content, that can be viewed by the user. A set of tools – the navigation service, navigation tag library and navigation iViews – enable you to display the navigation tree for the current user and provide the necessary links for navigation.

This section provides background on how portal navigation works and describes the main tasks for managing portal navigation. This section contains the following:

- [Architecture \[Page 33\]](#)
- [Creating Navigating iViews \[Page 33\]](#)
- [Creating Navigation Connectors \[Page 33\]](#)
- [Triggering Navigation \[Page 33\]](#)

3.7.1.1 Architecture

This section describes how navigation is handled in the portal, and includes the following:

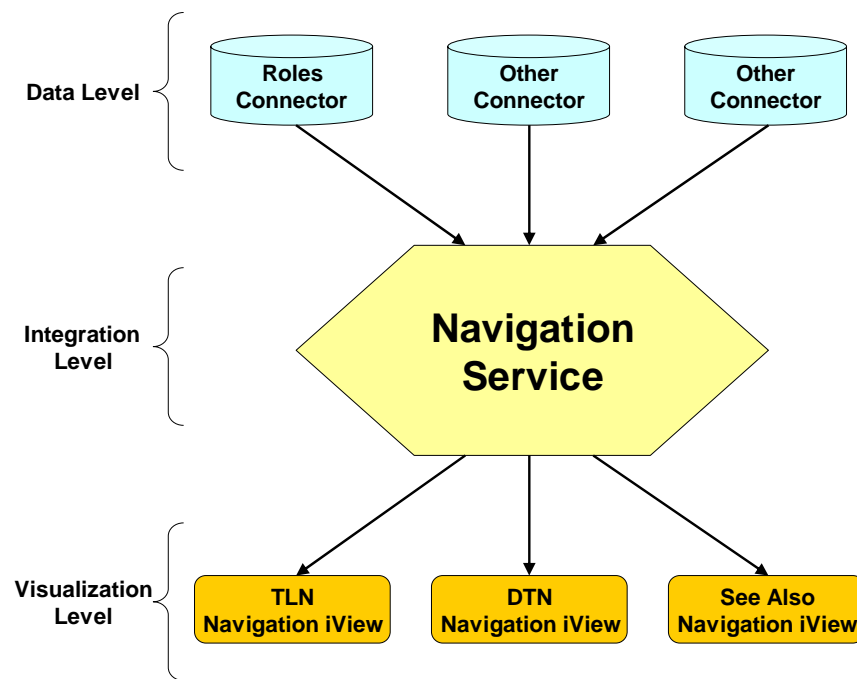
- [Navigation Model \[Page 33\]](#): Describes the major components that create and display the navigation hierarchy.
- [Navigation Hierarchy \[Page 33\]](#): Describes the types of nodes within the navigation hierarchy.
- [Navigation URL \[Page 33\]](#): Describes the URL for navigating to a specific navigation node.

3.7.1.1.1 Navigation Model

Navigation is based on the following three-level model:

- **Data Level:** Contains the navigation connectors that generate the navigation nodes and the navigation hierarchy. The portal comes with a roles connector, and you can create additional connectors in order to create additional navigation nodes and hierarchies.
- **Integration Level:** Retrieves navigation nodes and hierarchies from the connectors and creates the final navigation hierarchy for each user. Also provides services, such as the look up of navigation nodes.
- **Visualization Level:** Contains navigation iViews for displaying the navigation hierarchy for the current user and providing navigation links.

Modifying the Desktop and Navigation



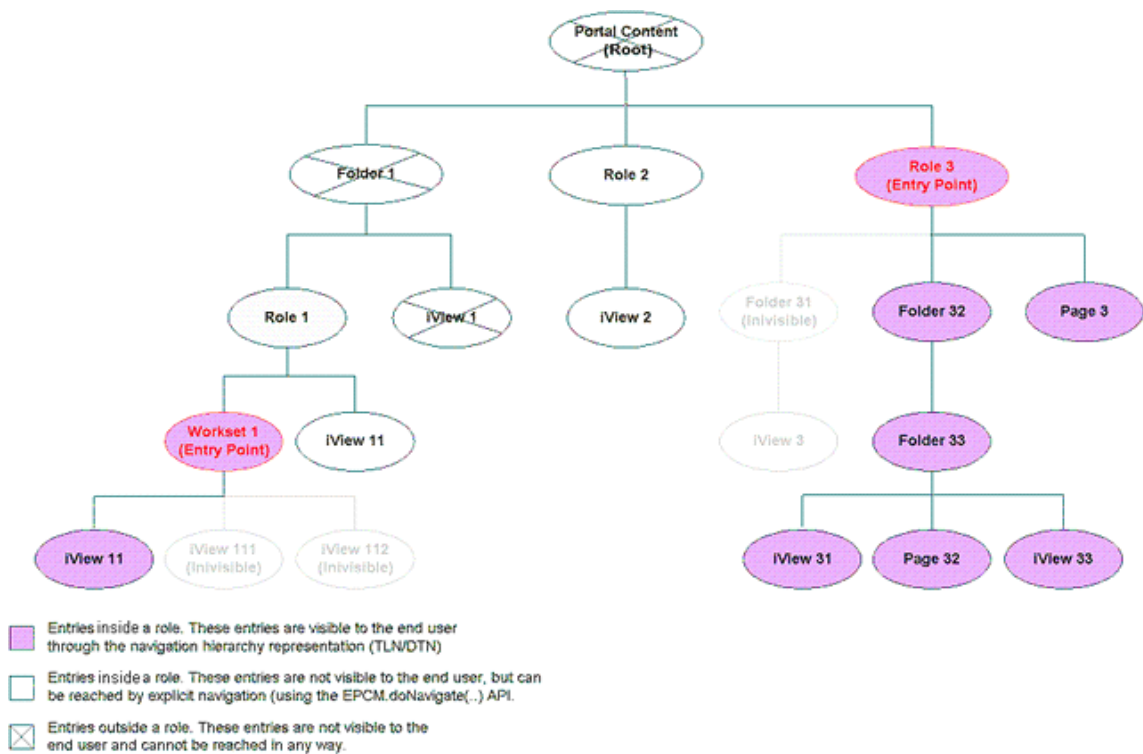
Navigation data is retrieved as follows:

1. The portal initializes the navigation service.
2. Navigation connectors are registered with the navigation service.
3. Navigation iViews call the navigation service to retrieve the navigation nodes, either initial nodes or specific nodes.
4. The navigation service retrieves the navigation information from the registered navigation connectors and returns navigation nodes to the navigation iViews. The navigation service preprocesses the request by sorting and merging the nodes.
5. Navigation iViews use the retrieved navigation nodes to display the navigation hierarchy.

3.7.1.1.2 Navigation Hierarchy

The navigation hierarchy is a structure of navigation nodes. It can be maintained in a tree, map, list or array structure or any other structure that organizes the nodes correctly. The following diagram shows an example of a navigation hierarchy using a tree structure and based on portal roles:

Modifying the Desktop and Navigation



The diagram shows the different types of nodes:

- **Valid/Visible:** For these nodes, navigation iViews display a link to the content. This type of node is located under one of the initial nodes in the navigation hierarchy.
- **Valid/Invisible:** For these nodes, navigation iViews do not display a link to the content, but the user is still allowed to view the content. The user can access the content by supplying the correct navigation URL or by clicking a link supplied with a content iView.

A navigation node is valid/invisible when:

- It is located under a role but not under an initial node in the navigation hierarchy.
 - Its `invisible` attribute is set to `true`.
- **Invalid:** The user is not allowed to access the content for this node.

3.7.1.1.3 Navigation URLs

A URL that navigates to a specific navigation node is made up of the base portal URL, plus a `NavigationTarget` parameter, whose value is a navigation target.

For example, the following URL

```
http://myServer:50000/irj/portal?
  NavigationTarget=ROLES://portal_content/myFolder/myRole
```


Modifying the Desktop and Navigation

navigates to the navigation node represented by the navigation target

`ROLES://portal_content/myFolder/myRole.`

The navigation target is made up of the following parts:

- **Prefix:** A string that represents the navigation connector that defines the requested navigation node.
For the roles connector, the prefix is `ROLES.`
- **Separator:** The character string `://`
- **Internal URL or Path:** A string that can be interpreted by the specified navigation connector.

In the example, the path is `portal_content/myFolder/myRole.`

3.7.1.1.4 Navigation Service

The navigation service enables you to retrieve information about the navigation hierarchy for a specific user.

The service can be obtained with the following code:

```
INavigationService service = (INavigationService)
PortalRuntime.getRuntimeResources().getService(INavigationService.KEY);
```

The service provides the following methods:

- **getInitialNodes():** Queries the navigation connectors and returns all the top-level nodes.
- **getFirstNode():** Returns the first node in the navigation hierarchy, if it is launchable. If the first node is not launchable, the method returns the first child of the first node in the navigation hierarchy.
- **getNode():** Returns the `INavigationNode` for a specified navigation target.
- **getNodes():** Returns a `Vector` of `INavigationNode` objects for a specified `Vector` of navigation targets.
- **getNodeByQuickLink():** Returns the navigation node for a specified quick link.
- **getNavigationNodeHashedName():** Takes the long name for a navigation node and returns the node's hashed name.
- **getNavigationNodeOriginalName():** Takes the hashed name for a navigation node and returns the node's long name.

All of the navigation service methods require you to pass a `Hashtable` of environment variables, one of which must be an `IUser` object for the current user. This value must be stored with the key `NavigationPrincipal`; a constant is defined for this value in

`INavigationConstants.`



Note that navigation connectors supply `INavigationConnectorNode` (`AbstractNavigationConnectorNode`) objects to the navigation service, while the navigation service supplies `INavigationNode` objects to navigation `iViews` (and all other components that call the service).

Additional Interfaces

The navigation service also implements the following interfaces, which provide additional methods for working with the navigation hierarchy:

- **INavigationConnectorRegistration:** Provides methods for registering a navigation connector and redirector.
For more information, see [Creating Navigation Connectors \[Page 33\]](#).
- **INavigationNamingHandler:** Provides methods for composing navigation node names.
- **INavigationRedirector:** Provides a method to retrieve redirected node names. A redirector must be registered that recognizes the prefix of the navigation target.
For more information, see [Redirectors \[Page 33\]](#).
- **INavigationGenerator:** Provides methods for creating the JavaScript (such as the `doNavigate()` method) for navigation to a navigation target.
For more information on triggering navigation, see [Triggering Navigation \[Page 33\]](#) and [Client-Side Eventing \[Page 33\]](#).
- **IObjectBasedNavigation:** Provides methods for creating object-based navigation links.
For more information, see [Object-Based Navigation \[Page 33\]](#).

To obtain an object that implements one of these interfaces, get the navigation service (with `INavigationService.Key`) and cast it into the desired interface.

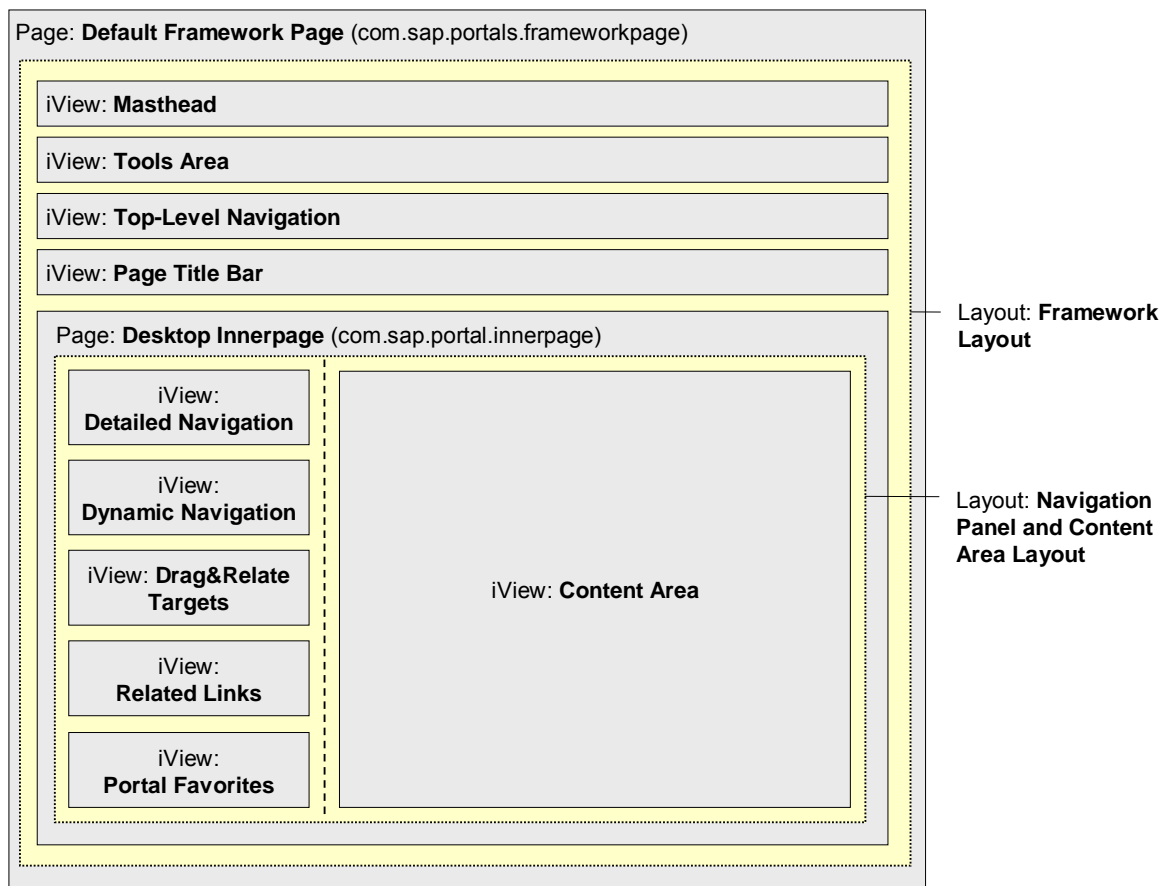
For example:

```
INavigationNamingHandler service = (INavigationNamingHandler)
PortalRuntime.getRuntimeResources().getService(INavigationService.KEY
);
```

3.7.1.1.5 Framework Page

The following shows the default layout for the default framework page (the names in parentheses are the PCD IDs for these objects):

Modifying the Desktop and Navigation



A navigation iView can be content independent (always visible, for example, the top-level navigation iView) or content dependent (only visible sometimes, for example, the *See Also* panel, which is displayed only when related links are defined for the current content).

The following are the default iViews within the framework page.

- **Masthead:** Displays the personalized welcome message and portal links, such as *Help*, *Personalize* and *Log Off*.
- **Tools Area:** Contains a search input field for Knowledge Management (KM) components. Except for KM enabled installations, this iView is invisible. A portal administrator can edit the portal framework page and enable the iView.
- **Top Level Navigation (TLN):** Displays the top levels of the navigation hierarchy for the current user, by default two levels. The following are key properties of the TLN:
 - **Hovering:** In hovering mode, the user can hover over a first-level node and this node's second-level nodes are displayed.
 - **Number of Display Levels:** By default, the TLN shows two root levels of the navigation hierarchy. If this attribute is 0, the TLN is not shown, and the hierarchy is displayed in the DTN. If this attribute is 1, the TLN contains one level and the rest of the navigation hierarchy is displayed in the DTN.
 - **Personalize Portal Page:** The PCD location of the personalization portal page. By default, the portal personalization dialog page is used.

Modifying the Desktop and Navigation

- **Page Title Bar:** Displays links for page-related functions, such as refreshing the page or opening the page in a new window.
- **Detailed Navigation (DTN):** Displays the portal navigation hierarchy, by default starting from the third level.

The first level to be displayed in the DTN iView depends on what is displayed in the TLN. For example, if the TLN displays one level, the DTN should start from the second level. The `Start At Level` attribute specifies the start level in the DTN.

- **Dynamic Navigation:** Displays navigation iViews assigned to the current content.
- **Drag&Relate Targets:** Displays a list of iViews related to the current content and for which Drag&Relate actions can be performed. A content administrator can select iViews and pages to associate to the current content, and Drag&Relate links to these iViews and pages are displayed in the Drag&Relate iView.

For more information on Drag&Relate actions, see *Drag&Relate Targets* in the *Portal Administration Guide*.

- **See Also (Related Links):** Displays a list of iViews related to the current content. A content administrator can select iViews and pages to associate to the current content, and links to these iViews and pages are displayed in the See Also iView.
- **Portal Favorites:** Displays a list of links to pages and iViews that the user has chosen as favorites.

Location of Components

The following lists the portal applications that contain the framework page, navigation iViews and layouts:

- **Default Framework Page**

Component	Portal Application
Default Framework Page	<code>com.sap.portal.layouts.framework (framework.jsp)</code>
Innerpage	<code>com.sap.portal.layouts.framework (WAandNavPanel.jsp)</code>
Navigation Area	<code>com.sap.portal.layouts.framework (dynNavArea.jsp)</code>
Content Area	<code>com.sap.portal.navigation.contentarea</code>

- **Navigation iViews:**

Component	Portal Application
Masthead	<code>com.sap.portal.navigation.masthead</code>
Tools Area	<code>com.sap.portal.navigation.toolarea</code> (only delivered with Knowledge Management)
TLN	<code>com.sap.portal.navigation.toplevel</code>
Page Title Bar	<code>com.sap.portal.navigation.pagetoolbar</code>
DTN	<code>com.sap.portal.navigation.detailedtree</code>
Dynamic Navigation	<code>com.sap.portal.navigation.dynnavarea</code>
Drag&Relate Targets	<code>com.sap.portal.navigation.targets</code> (same iView as Related Links)
See Also (Related Links)	<code>com.sap.portal.navigation.targets</code> (same iView as Drag&Relate)

Modifying the Desktop and Navigation

Portal Favorites	<code>com.sap.km.cm</code> (only delivered with Knowledge Management)
------------------	--

3.7.1.1.5.1 Light Framework Page

The portal provides an additional framework page for displaying light content for an external-facing portal. Light content is content that uses fewer resources than standard content, for example, because it does not use HTMLB or client-side eventing Javascript.

The light framework page differs from the default framework page in the following ways:

- The light framework page creates a single-frame page, avoiding the need for client-side Javascript that would enable different frames to communicate with each other.
- The light framework page uses light navigation iViews, which avoid the use of resource-heavy HTMLB and client-side eventing Javascript.
- For more information about an external-facing portal, see [Implementing an External-Facing Portal \[External\]](#).

Location of Components

The following additional components are provided for the light framework page:

- **Default Framework Page**

Component	Portal Application
Light Framework Page	<code>com.sap.portal.layouts.framework</code> (light_framework.jsp)
Light Innerpage	<code>com.sap.portal.layouts.framework</code> (light_WAandNavPanel.jsp)
Light Content Area	<code>com.sap.portal.navigation.contentarea</code>

The Navigation Area component for the light framework page is the same as the one in the default framework page.

- **Navigation iViews:**

Component	Portal Application
Light Masthead	<code>com.sap.portal.navigation.masthead</code>
Light TLN	<code>com.sap.portal.navigation.lighttoplevel</code>
Light Page Title Bar	<code>com.sap.portal.navigation.pagetoolbar</code>
Light DTN	<code>com.sap.portal.navigation</code> <code>.lightdetailednavigationtree</code>

The Tools Area, Dynamic Navigation, Drag&Relate Targets, See Also (Related Links) and Portal Favorites iViews are the same as those in the default framework page.

3.7.1.1.6 Special Navigation Features

This section describes the following special features of portal navigation:

- [Navigation Cache \[Page 33\]](#)
- [Short \(Hashed\) URLs \[Page 33\]](#)
- [Quick Links \[Page 33\]](#)

Modifying the Desktop and Navigation

3.7.1.1.6.1 Navigation Cache

The portal caches the set of navigation nodes that is returned for each user. If another user is entitled to the exact same set of navigation nodes, the navigation service retrieves this set of navigation nodes from the cache instead of creating them again.

Each set of nodes is cached with a unique key, or discriminator. For the roles connector, each set of nodes is given a key based on the combination of roles that created the set of nodes. For example, for the set of navigation nodes for the Content Administrator and User Administrator roles, the nodes in these roles are cached together with a key that indicates this combination of roles, perhaps `ContentAdmin/SystemAdmin`.

The following is an example process flow for navigation caching:

1. User A with the roles Content Administrator and System Administrator logs on.
2. The portal generates the nodes for these roles, and then saves them in the cache with a key `ContentAdmin/SystemAdmin`.
3. User B with the role Content Administrator logs on.
4. The portal generates the nodes for this role, and then saves them in the cache with a key `ContentAdmin`. The portal does not retrieve them from the cache because there is no set of nodes in the cache for someone with just the Content Administrator role.
5. User C with the roles Content Administrator and System Administrator logs on.
6. The portal retrieves the nodes for these roles from the cache.

The cache is maintained on the server, and is turned off by default.



If you modify a navigation connector and redeploy it, you should clear the navigation cache.

For information on enabling or clearing the navigation cache, see [Navigation Cache \[External\]](#) in the *Portal Administration Guide*.

Custom Navigation Connectors

To support navigation caching, a custom navigation connector must implement `getConnectorCacheDiscriminator()`, which returns a key for the set of navigation nodes for the current user.

The following is the signature for `getConnectorCacheDiscriminator()`:

```
public String getConnectorCacheDiscriminator(Hashtable environment);
```

You may want to create the navigation nodes – and, therefore, the cache discriminator – based on the current user. You can retrieve the `IUser` object for the current user from the `Hashtable` passed into the method, as follows:

```
IUser user = (IUser)
    environment.get(INavigationConstants.NAVIGATION_PRINCIPAL);
```

The final discriminator is a combination of the discriminators generated by all navigation connectors for a user's set of navigation nodes.

Modifying the Desktop and Navigation

3.7.1.1.6.2 Short (Hashed) URLs

The portal supports navigation based on short (hashed) URLs. Instead of a URL such as the following:

```
http://myServer:50000/irj/portal?NavigationTarget=ROLES://portal_content/administrator/super_admin/super_admin_role/com.sap.portal.system_administration/com.sap.portal.system_admin_ws/com.sap.portal.permissions
```

the portal creates another URL for the same navigation target such as the following:

```
http://myServer:50000/irj/portal?NavigationTarget=navurl://0c3c7ac0dfe1083d8f50ae954b8ec25f
```

An administrator can turn this feature on, and then the navigation service creates hashed URLs for all navigation nodes. All links created by the navigation service are generated as hashed URLs and not as the original URL. For more information on links, see [Navigation URLs \[Page 33\]](#).



You do not have to do anything to have the navigation service create hashed URLs for your connector. If the feature is turned on by an administrator, the hashing occurs automatically.

You can call the following methods to get the long and hashed URL of a navigation node:

- **getHashed()**: Returns the short (hashed) URL. If the hashed URL feature is turned off, this method returns the long URL.
- **getName()**: Returns the long URL.

By default, this feature is on.

3.7.1.1.6.3 Quick Links

The portal enables users to navigate to a specific node by entering the portal's base address followed by a short string, or quick link.

For nodes defined by the roles connector, a portal administrator can assign quick links to a node by setting the `Quick Link` property of the `iView` or page associated with the node.

For nodes defined by a custom navigation connector, the connector must implement the method `getNodeByQuickLink()`, which determines what node to return for a specific quick link. For example, you can keep a `Map` of quick link strings and its corresponding node.

The following is the signature for `getNodeByQuickLink()`:

```
public NamingEnumeration getNodeByQuickLink(
    Hashtable environment, String quickLink);
```

Quick links must be composed of only URI unreserved characters, which are letters, digits, hyphen (-), period (.), underscore (_) and tilde (~).

3.7.1.2 Creating Navigation iViews

Navigation iViews are the iViews within the framework page that provide links for navigating to different content in the portal.

The portal comes with a default framework page that includes a set of default navigation iViews.

The default iViews can also be configured by a portal administrator to change the look and feel, without the need for rewriting the iView. For example, an administrator can configure the masthead to display the *New Session* link or the page title bar to hide the *History* link. For more information, see *Navigation* in the *Portal Administration Guide*.

You can customize the look and feel of the portal by creating new navigation iViews, and then replacing the default iViews with your iViews or simply adding your iViews to the framework page.

The recommended way to create navigation iViews is by writing a JSP page and using built-in tag libraries: [Navigation Tag Library \[Page 33\]](#) for iViews that display navigation nodes and [Framework Tag Library \[Page 33\]](#) for masthead and page title bar iViews.

For more information on creating JSP pages in the portal, see [Writing JSP Pages \[Page 33\]](#).

For more information on the default navigation iViews, see [Framework Page \[Page 33\]](#).

3.7.1.2.1 Navigation Tag Library

Purpose

The navigation tag library enables you to easily develop navigation iViews based on JSP pages that use the tag library.

The tags provide access to the navigation nodes for the current user, and enable you to iterate through those nodes. You can create top-level and detailed navigation iViews by iterating through the navigation nodes and building and displaying navigation links.



The tag library examples in this chapter assume that the navigation iView is placed within a light framework page, which provides its own style sheet with unique styles for light pages.

This section contains the following:

- [Types of Tags \[Page 33\]](#)
- [How to Use the Tag Library \[Page 33\]](#)
- [Tag Reference \[Page 33\]](#)
- [Samples \[Page 33\]](#)

Prerequisites

- You are familiar with how to write JSP pages in the portal, as described in [Writing JSP Pages](#).

3.7.1.2.1.1 Types of Tags

The navigation tag library contains the following types of tags:

- **Iterator:** Iterates through a set of nodes, exposing in its body a different navigation node during each iteration.

Modifying the Desktop and Navigation

- **Node:** Exposes in its body a specific navigation node.
- **Operator:** Performs an operation, such as storing a navigation node for later recall.
- **Rendering:** Outputs HTML based on the currently exposed navigation node.
- **Conditional:** Includes its body if a condition is true or false.

Iterator Tags

These tags enable you to iterate through a set of nodes, for example the top-level navigation nodes or the children of a node.

In the body of each tag, a different navigation node is exposed during each iteration. You can include cooperating tags – for example, `navNodeAnchor` for building a link from the current node – that work with the current node in the iteration.

The navigation tag library includes the following iterator tags:

- [iterateInitialNavNodes \[Page 33\]](#)
- [iterateNavNodeChildren \[Page 33\]](#)
- [iterateNavNodesInSelectedPath \[Page 33\]](#)
- [iterateSelectedNavNodesLevel \[Page 33\]](#)
- [recurseNavNodeChildren \[Page 33\]](#)

Node Tags

These tags enable you to select a specific navigation node, such as the currently selected node or the parent of the current node.

In the body of each tag, the specific navigation node is exposed.

The navigation tag library includes the following node tags:

- [launchedNavNode \[Page 33\]](#)
- [navNode \[Page 33\]](#)
- [navNodeParent \[Page 33\]](#)
- [selectedNavNode \[Page 33\]](#)

Operator Tags

These tags perform an operation, such as storing a node. Generally, these are closed tags.

The navigation tag library includes the following operator tags:

- [storeNavNode \[Page 33\]](#)
- [recallNavNode \[Page 33\]](#)
- [doNotRecurseNavNodeChildren \[Page 33\]](#)

Rendering Tags

These tags output HTML to be displayed in the iView.

The navigation tag library includes the following rendering tags:

- [navNodeAnchor \[Page 33\]](#)
- [navNodeDescription \[Page 33\]](#)

Modifying the Desktop and Navigation

- [navNodePictogram \[Page 33\]](#)
- [navNodeTitle \[Page 33\]](#)

Conditional Tags

These tags enable you to include HTML or perform some action based on the currently exposed navigation node or the current user. A tag's body is included if the condition for that tag is true.

For example, you can check whether the current node is an iView node and, if it is, include an iView icon in the detailed navigation tree, as shown in the following:

```
<nav:selectedNavNode>
  <nav:ifNavNodeIsIView>
    ... include an icon ...
  </nav:ifNavNodeIsIView>
</nav:selectedNavNode>
```

The navigation tag library includes the following conditional tags:

- [ifAnonymousUser \[Page 33\]](#)
- [ifHasMoreIterations \[Page 33\]](#)
- [ifNavNodeEqualsLaunchedNavNode \[Page 33\]](#)
- [ifNavNodeEqualsSelectedNavNode \[Page 33\]](#)
- [ifNavNodeHasChildren \[Page 33\]](#)
- [ifNavNodeInSelectedPath \[Page 33\]](#)
- [ifNavNodeIsFolder \[Page 33\]](#)
- [ifNavNodeIsIView \[Page 33\]](#)
- [ifNavNodeIsPage \[Page 33\]](#)
- [ifNavNodeVisualizationType \[Page 33\]](#)
- [ifNextRecursionDepthWillDecrease \[Page 33\]](#)
- [ifNextRecursionDepthWillIncrease \[Page 33\]](#)
- [ifNextRecursionDepthWillNotChange \[Page 33\]](#)

For most of the tags above, the tag library includes a second tag for checking whether the condition is false. For more information on these tags, see [ifNot ... \[Page 33\]](#).

3.7.1.2.1.1 Glossary

The following are key terms for describing the navigation tag library:

- **Current Node:** In an iterator tag, the node that is exposed during the current iteration.
- **Selected Node:** The node that the end user selected by clicking a navigation link, generally within a navigation iView.
- **Nodes in the Selected Path:** The set of nodes in the current user's navigation tree that form the path from the initial nodes to the selected nodes.

For example, if the user selects *Content Administration* → *Users* → *Create User*, all three nodes are in the selected path, and *Create User* is the selected node.

Modifying the Desktop and Navigation

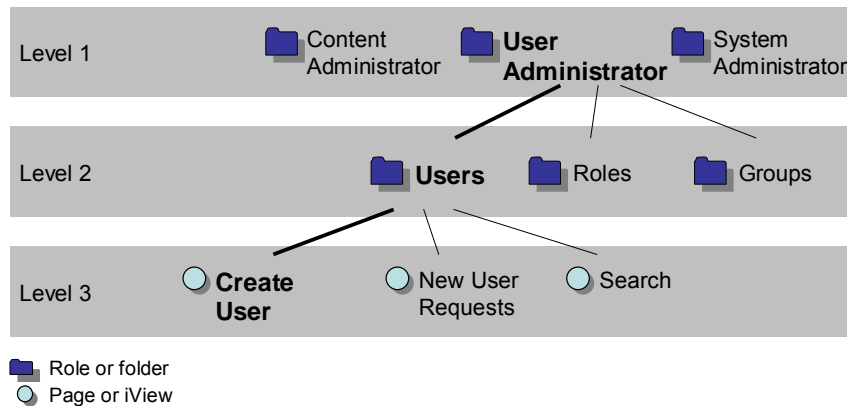
The following tags work with the nodes in the selected path:

- [iterateNavNodesInSelectedPath \[Page 33\]](#): Iterates through the nodes in the selected path.
In the example above, the tag would iterate three times, first exposing the *Content Administration* node, then the *Users* node, and then the *Create Users* node.
- [iterateSelectedNavNodesLevel \[Page 33\]](#): Iterates through the siblings of the node in the selected path on a specified level.
In the example above, if level 2 is specified, then the tag would iterate through the children nodes of the *Content Administration* node (that is, the *Users* node and its siblings).

- **Launched Node:** The node that was launched because of the most recent navigation.
Generally, the launched node is the same as the selected node, but can be different. For example, if the user selects a folder node, the launched node is the first child node of the folder that is a page or iView.

Example

The following is an example of a partial navigation tree for the super admin role.



- If the user clicks the *Create User* link, the *Create User* node is the selected node, as well as the launched node.
The nodes shown in **bold** are the nodes in the selected path. The `iterateNavNodeInSelectedPath` tag iterates through these nodes. The `iterateSelectedNavNodesLevel` tag, with level 2 specified, iterates through the Level 2 nodes displayed (which are the children of the *User Administration* node).
- If the user clicks the *Users* link, the *Users* node is the selected node, and the *Create User* node is the launched node (because it is the first page or iView child of the *Users* node).

Modifying the Desktop and Navigation

3.7.1.2.1.2 How to Use the Tag Library

The navigation tag library is designed to enable you to build navigation iViews based on JSP pages, which are deployed to the portal as portal components in a PAR file that contains the JSP page and a `portalapp.xml` file.

For more information on writing JSP pages for the portal, see [Writing JSP Pages \[Page 33\]](#).

Referencing the Tag Library

To use navigation tags in a JSP page, do the following:

1. At the top of the JSP page, add a reference to the tag library, as follows:

```
<%@ taglib uri="NavigationTagLibrary" prefix="nav" %>
```

2. In the `portalapp.xml` file:

- a. Add a sharing reference in the `<application-config>` element, as follows:

```
<property name="SharingReference"
  value="com.sap.portal.navigation.navigationtaglibrary"/>
```

- b. Add a reference to the tag library by adding the following property to the `<component-profile>` element for the JSP component:

```
<property name="NavigationTagLibrary"
  value="/SERVICE/com.sap.portal.navigation.navigationtaglibrary/
taglib/TagLibrary.tld"/>
```

3.7.1.2.1.3 Tag Reference

This section lists all the tags in the navigation tag library.

3.7.1.2.1.3.1 doNotRecurseNavNodeChildren

Indicates that the children of the current node should not be included in the recursive iteration in a `recurseNavNodeChildren` tag.

Cooperating Tags

The tag must be nested in the following tag:

- `recurseNavNodeChildren`

Example

The following creates a detailed navigation iView and displays the navigation tree starting at a specified level. Each node can be displayed as either open (its children are also displayed) or closed (its children are not displayed).

The `recurseNavNodeChildren` tag starts a depth-first traversal through all nodes below the current node. If a node is closed, its children are not displayed. The `doNotRecurseNavNodeChildren` tag is used to skip the traversal through the children nodes of the current node and prevent these children nodes from being displayed.

```
<TABLE Width='100%' Class="lightDTNTable">
  <nav:iterateSelectedNavNodesLevel level="<%=strStartLevel%>"
    currentNavNode="currentRoot">
    ...
  </nav:iterateSelectedNavNodesLevel>
</TABLE>
```

Modifying the Desktop and Navigation

```

<nav:recurseNavNodeChildren currentDepth="currentDepth"
  currentNavNode="currentNavNode">
  <%
      ... Determine if node has children

      if(!nodeIsOpen) {
          %><nav:doNotRecurseNavNodeChildren/><%
      } %>
      <TR Style="{text-indent:
          ... Display Node

      <TR/>
  </nav:recurseNavNodeChildren>
</nav:iterateSelectedNavNodesLevel>
</TABLE>

```

See Also

- [recurseNavNodeChildren \[Page 33\]](#)

3.7.1.2.1.3.2 ifAnonymousUser

Includes its body if the current user is an anonymous user.

Example

The following displays a link to a logon page if the use is anonymous, and a welcome greeting if the user is not anonymous.

```

<nav:ifAnonymousUser>
    ... Display logon link
</nav:ifAnonymousUser>

<nav:ifNotAnonymousUser>
    ... Display greeting for current user
</nav:ifNotAnonymousUser>

```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.3 ifHasMoreIterations

Includes its body if the current node is not the last node in the current iteration.

Modifying the Desktop and Navigation

For example, if you display separators between the names of nodes, such as in a top-level navigation `iView`, use this tag to display the separator and the `ifNotHasMoreIterations` tag to prevent a separator after the last node.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `recurseNavNodeChildren`

Example

The following displays a separator if the iteration has more nodes.

```
<nav:ifHasMoreIterations>
  <TD nowrap class="spacingTDPipeLevel2"> | </TD>
</nav:ifHasMoreIterations>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.4 ifNavNodeEqualsLaunchedNavNode

Includes its body if the current node is the node that was launched during the most recent navigation.

For example, when creating a detailed navigation `iView`, you may want to bold the navigation node that is currently launched. You can add code to bold the current node in the `ifNavNodeEqualsLaunchedNavNode` tag during an iteration through all the navigation nodes.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`

Modifying the Desktop and Navigation

- `selectedNavNode`

Example

The following displays the current node in the iteration with a different style depending on whether the node is the launched node.

```
<nav:ifNavNodeEqualsLaunchedNavNode>
  <nav:navNodeAnchor navigationMethod="byURL"
    urlParameters="<%=urlParameters%>"
    anchorAttributes="class='lightDTNTextSelected'"/>
</nav:ifNavNodeEqualsLaunchedNavNode>
<nav:ifNotNavNodeEqualsLaunchedNavNode>
  <nav:navNodeAnchor navigationMethod="byURL"
    urlParameters="<%=urlParameters%>"
    anchorAttributes="class='lightDTNText'"/>
</nav:ifNotNavNodeEqualsLaunchedNavNode>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.5 ifNavNodeEqualsSelectedNavNode

Includes its body if the current node is the node that was selected by the user during the most recent navigation.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

See Also

- [ifNot ... \[Page 33\]](#)

Modifying the Desktop and Navigation

3.7.1.2.1.3.6 ifNavNodeHasChildren

Includes its body if the current node has children nodes.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays an icon and link to open and close the node if the node has children.

```
<nav:ifNavNodeIsFolder>
  <nav:ifNavNodeHasChildren>
    <A href="<%=componentURI%>?<%=urlParameters%>"
      class="lightDTNKnob
      <% if(nodeIsOpen) { %>Open<% } else { %>Closed<% } %>"/>
  </nav:ifNavNodeHasChildren>
  <nav:ifNotNavNodeHasChildren>
    <A class="lightDTNKnobNone"/>
  </nav:ifNotNavNodeHasChildren>
  <A class="lightDTN
    <% if(nodeIsOpen) { %>Open<% } else { %>Closed<% } %>Folder"/>
</nav:ifNavNodeIsFolder>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.7 ifNavNodeInSelectedPath

Includes its body if the current node is one of the nodes in the path of selected nodes.

The node is in the selected path if it is the selected node, or the parent of the selected node, or the grandparent of the selected node, and so forth. For more information about the selected path, see [Glossary \[Page 33\]](#).

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`

Modifying the Desktop and Navigation

- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays the top-level navigation nodes and displays in a special style the initial node in the selected path.

```
<nav:iterateInitialNavNodes>
  <nav:ifNavNodeInSelectedPath>
    <TD nowrap class="chosenTDLevel1">
      <nav:navNodeAnchor navigationMethod="byURL"
        anchorAttributes="class='chosenOnLevel1'"/>
    </TD>
    <% if(levels==2) { %>
      <nav:storeNavNode/>
    <% } %>
  </nav:ifNavNodeInSelectedPath>
  <nav:ifNotNavNodeInSelectedPath>
    <TD nowrap class="unChosenTDLevel1">
      <nav:navNodeAnchor navigationMethod="byURL"
        anchorAttributes="class='unChosenOnLevel1'"/>
    </TD>
  </nav:ifNotNavNodeInSelectedPath>
</nav:iterateInitialNavNodes>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.8 ifNavNodeIsFolder

Includes its body if the current node's visualization type is folder.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`

Modifying the Desktop and Navigation

- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays a folder icon (either opened or closed) if the current node is a folder node.

```
<nav:ifNavNodeIsFolder>
  <nav:ifNavNodeHasChildren>
    <A href="<%=componentURI%>?<%=urlParameters%>"
      class="lightDTNKnob<% if(nodeIsOpen) { %>Open<% }
        else { %>Closed<% } %>" />
  </nav:ifNavNodeHasChildren>
  <nav:ifNotNavNodeHasChildren>
    <A class="lightDTNKnobNone" />
  </nav:ifNotNavNodeHasChildren>
  <A class="lightDTN<% if(nodeIsOpen) { %>Open<% }
    else { %>Closed<% } %>Folder" />
</nav:ifNavNodeIsFolder>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.9 ifNavNodeIsIView

Includes its body if the current node's visualization type is iView.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays an iView icon if the current node is an iView.

Modifying the Desktop and Navigation

```
<nav:ifNavNodeIsIView>

... Display iView icon

</nav:ifNavNodeIsIView>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.10 ifNavNodeIsPage

Includes its body if the current node's visualization type is page.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays a page icon if the current node is a page.

```
<nav:ifNavNodeIsPage>

... Display page icon

</nav:ifNavNodeIsPage>
```

See Also

- [ifNot ... \[Page 33\]](#)

3.7.1.2.1.3.11 ifNavNodeVisualizationType

Includes its body if the current node is of the type specified by the `equals` attribute.

The tag library includes tags for each individual type of node, for example, `ifNavNodeIsIView`. This tag enables the development of additional types without the need to create new tags.

Attributes

Name	Mandatory	Description
<code>equals</code>	Yes	<p>A constant that represents a type of navigation node. The constants for this attribute are defined in the <code>INavigationConstants</code> interface of the <code>com.sapportals.portal.navigation</code> package.</p> <p>The following constants are defined:</p> <ul style="list-style-type: none">• <code>TYPE_IVIEW</code>• <code>TYPE_PAGE</code>• <code>TYPE_FOLDER</code> <p><code>INavigationConstants</code> also provides the constant <code>TYPE_WORKSET</code> and <code>TYPE_OTHER</code>, but these are not valid for this tag.</p>

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

```
<nav:ifNavNodeVisualizationType equals="1">
... Display icon for visualization type
</nav:ifNavNodeVisualizationType>
```

See Also

- [ifNot ... \[Page 33\]](#)
- [ifNavNodesIsFolder \[Page 33\]](#)
- [ifNavNodesIsView \[Page 33\]](#)
- [ifNavNodesIsPage \[Page 33\]](#)

Modifying the Desktop and Navigation

3.7.1.2.1.3.12 ifNextRecursionDepthWillDecrease

Includes its body if the next node in a recursive iteration is a node at a higher level than the current node – that is, there are more nodes and the current node has no children or any more siblings.

Cooperating Tags

The tag must be nested in the following tag:

- `recurseNavNodeChildren`

Example

The following displays images for building a visual tree of the navigation nodes.

```
<nav:recurseNavNodeChildren>
  <nav: ifNextRecursionDepthWillNotChange >

    ... Display image for node with no children but with more
    siblings

  </nav:ifNextRecursionDepthWillNotChange>

  <nav: ifNextRecursionDepthWillIncrease >

    ... Display image for a node with children

  </nav:ifNextRecursionDepthWillIncrease>

  <nav:ifNextRecursionDepthWillDecrease>

    ... Display image for a node with no children and no more
    siblings

  </nav:ifNextRecursionDepthWillDecrease>
</nav:recurseNavNodeChildren>
```

See Also

- [ifNextRecursionDepthWillIncrease \[Page 33\]](#)
- [ifNextRecursionDepthWillNotChange \[Page 33\]](#)

3.7.1.2.1.3.13 ifNextRecursionDepthWillIncrease

Includes its body if the next node in a recursive iteration is a child of the current node.

Cooperating Tags

The tag must be nested in the following tag:

- `recurseNavNodeChildren`

Modifying the Desktop and Navigation

Example

The following displays images for building a visual tree of the navigation nodes.

```
<nav:recurseNavNodeChildren>
  <nav: ifNextRecursionDepthWillNotChange >

    ... Display image for node with no children but with more
    siblings

  </nav:ifNextRecursionDepthWillNotChange>

  <nav: ifNextRecursionDepthWillIncrease >

    ... Display image for a node with children

  </nav:ifNextRecursionDepthWillIncrease>

  <nav:ifNextRecursionDepthWillDecrease>

    ... Display image for a node with no children and no more
    siblings

  </nav:ifNextRecursionDepthWillDecrease>
</nav:recurseNavNodeChildren>
```

See Also

- [ifNextRecursionDepthWillDecrease \[Page 33\]](#)
- [ifNextRecursionDepthWillNotChange \[Page 33\]](#)

3.7.1.2.1.3.14 ifNextRecursionDepthWillNotChange

Includes its body if the next node in a recursive iteration is a sibling of the current node.

Cooperating Tags

The tag must be nested in the following tag:

- `recurseNavNodeChildren`

Example

The following displays images for building a visual tree of the navigation nodes.

Modifying the Desktop and Navigation

```

<nav:recurseNavNodeChildren>
  <nav: ifNextRecursionDepthWillNotChange >

    ... Display image for node with no children but with more
    siblings

  </nav:ifNextRecursionDepthWillNotChange>

  <nav: ifNextRecursionDepthWillIncrease >

    ... Display image for a node with children

  </nav:ifNextRecursionDepthWillIncrease>

  <nav:ifNextRecursionDepthWillDecrease>

    ... Display image for a node with no children and no more
    siblings

  </nav:ifNextRecursionDepthWillDecrease>
</nav:recurseNavNodeChildren>

```

See Also

- [ifNextRecursionDepthWillDecrease \[Page 33\]](#)
- [ifNextRecursionDepthWillIncrease \[Page 33\]](#)

3.7.1.2.1.3.15 ifNot ...

For many of the conditional tags, a second tag is included for testing whether the condition is false. If the condition is false, the tag's body is included.

Often, it is necessary to include HTML if a condition is true and to include different HTML if the condition is false. This is done by adding a conditional tag for testing if the condition is true, and then adding the corresponding tag for testing if the condition is false.

The following selects one style for an anchor if the current node is the launched node, and selects a different style if it is not the launched node:

```

<nav:ifNavNodeEqualsCurrentlyLaunchedNavNode>
  <nav:navNodeAnchor navigationMethod="byURL"
    urlParameters="<%=urlParameters%>"
    anchorAttributes="class='lightDTNTextSelected'"/>
</nav:ifNavNodeEqualsCurrentlyLaunchedNavNode>

<nav:ifNotNavNodeEqualsCurrentlyLaunchedNavNode>
  <nav:navNodeAnchor navigationMethod="byURL"
    urlParameters="<%=urlParameters%>"
    anchorAttributes="class='lightDTNText'"/>
</nav:ifNotNavNodeEqualsCurrentlyLaunchedNavNode>

```

The tags for checking if a condition is false are the same as the tags for checking if the condition is true, except that they start with `ifNot` instead of `if`.

The following is a list of tags for checking if a condition is false.

Tag	Description
-----	-------------

Modifying the Desktop and Navigation

<code>ifNotAnonymousUser</code>	Includes its body if the current user is not an anonymous user. See ifAnonymousUser [Page 33] .
<code>ifNotHasMoreIterations</code>	Includes its body if the current node is the last node in the current iteration. See ifHasMoreIterations [Page 33] .
<code>ifNotNavNodeEqualsLaunchedNavNode</code>	Includes its body if the current node is not the node that was launched during the most recent navigation. See ifNavNodeEqualsLaunchedNavNode [Page 33] .
<code>IfNotNavNodeEqualsSelectedNavNode</code>	Includes its body if the current node is not the node that was selected by the user. See ifNavNodeEqualsSelectedNavNode [Page 33] .
<code>ifNotNavNodeHasChildren</code>	Includes its body if the current node has no children nodes. See ifNavNodeHasChildren [Page 33] .
<code>ifNotNavNodeIsFolder</code>	Includes its body if the current node is not a folder. See ifNavNodeIsFolder [Page 33] .
<code>ifNotNavNodeIsIView</code>	Includes its body if the current node is not an iView. See ifNavNodeIsIView [Page 33] .
<code>ifNotNavNodeIsPage</code>	Includes its body if the current node is not a page. See ifNavNodeIsPage [Page 33] .
<code>ifNotNavNodeInSelectedPath</code>	Includes its body if the current node is not in the selected path. See ifNavNodeInSelectedPath [Page 33] .
<code>ifNotNavNodeVisualizationType</code>	Includes its body if the current node is not of the type specified by the <code>equals</code> attribute. See ifNavNodeVisualizationType [Page 33] .

3.7.1.2.1.3.16 iterateInitialNavNodes

Iterates through the top-level navigation nodes.

During each iteration of the tag, a different node is exposed in the body of the tag, either via a cooperating tag that accesses the current node in the iteration or the scriptlet variable defined by the `currentNavNode` attribute.

Attributes

Name	Mandatory	Description
------	-----------	-------------

Modifying the Desktop and Navigation

<code>currentNavNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the current node in the iteration. The variable is accessible in scriptlets in the body of the tag.
<code>direction</code>	No	Indicates in what direction to iterate. The attribute can have the following values: <ul style="list-style-type: none"> • forward (default): Iterate from the first to the last node. • backward: Iterate from the last to the first node.

Variables

Name	Scope	Description
<code><currentNavNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the current node in the iteration.

Example

The following iterates through all the top-level navigation nodes and creates a link for each one.

```
<nav:iterateInitialNavNodes>
  <nav:ifNavNodeSelected>
    <TD nowrap class="chosenTDLevel1">
      <nav:navNodeAnchor navigationMethod="byURL"
        anchorAttributes="class='chosenOnLevel1'"/>
    </TD>
    <% if(levels==2) { %>
      <nav:storeNavNode/>
    <% } %>
  </nav:ifNavNodeSelected>
  <nav:ifNotNavNodeSelected>
    <TD nowrap class="unChosenTDLevel1">
      <nav:navNodeAnchor navigationMethod="byURL"
        anchorAttributes="class='unChosenOnLevel1'"/>
    </TD>
  </nav:ifNotNavNodeSelected>
</nav:iterateInitialNavNodes>
```

3.7.1.2.1.3.17 iterateNavNodeChildren

Iterates through the children of the current navigation node.

During each iteration of the tag, a different node is exposed in the body of the tag, either via a cooperating tag that accesses the current node in the iteration or the scriptlet variable defined by the `currentNavNode` attribute.

Attributes

Name	Mandatory	Description
------	-----------	-------------

Modifying the Desktop and Navigation

<code>currentNavNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the current node in the iteration. The variable is accessible in scriptlets in the body of the tag
<code>direction</code>	No	Indicates in what direction to iterate. The attribute can have the following values: <ul style="list-style-type: none"> • forward (default): Iterate from the first to the last node. • backward: Iterate from the last to the first node.

Variables

Name	Scope	Description
<code><currentNavNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the current node in the iteration.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following recalls a navigation node (for example, the top-level node that was selected by the user) and iterates through the node's children.

For more information on recalling a node, see [recallNavNode \[Page 33\]](#).

```
<nav:recallNavNode>
  <nav:iterateNavNodeChildren>
    <nav:ifNavNodeSelected>
      <TD nowrap class="chosenTDLevel2">
        <nav:navNodeAnchor navigationMethod="byURL"
          anchorAttributes="class='chosenOnLevel2'"/>
      </TD>
    </nav:ifNavNodeSelected>
    <nav:ifNotNavNodeSelected>
      <TD nowrap class="unChosenTDLevel2">
```

Modifying the Desktop and Navigation

```

        <nav:navNodeAnchor navigationMethod="byURL"
            anchorAttributes="class='unChosenOnLevel2'"/>
    </TD>
</nav:ifNotNavNodeSelected>
<nav:ifHasMoreIterations>
    <TD nowrap class="spacingTDPipeLevel2"> | </TD>
</nav:ifHasMoreIterations>
</nav:iterateNavNodeChildren>
    <TD nowrap class="spacingTDLevel2">&nbsp;</TD>
</nav:recallNavNode>

```

3.7.1.2.1.3.18 iterateNavNodesInSelectedPath

Iterates through the nodes (one on each level) that were selected by the user.

For example, if the user selects *Content Administration* → *Users* → *Create User*, the tag iterates through these three nodes.

During each iteration of the tag, a different node is exposed in the body of the tag, either via a cooperating tag that accesses the current node in the iteration or the scriptlet variable defined by the `currentNavNode` attribute.

For more information about the selected path, see [Glossary \[Page 33\]](#).

Attributes

Name	Mandatory	Description
<code>currentNavNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the current node in the iteration. The variable is accessible in scriptlets in the body of the tag.
<code>direction</code>	No	Indicates in what direction to iterate. The attribute can have the following values: <ul style="list-style-type: none"> forward (default): Iterate from the first to the last node. backward: Iterate from the last to the first node.

Variables

Name	Scope	Description
<code><currentNavNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the current node in the iteration.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`

Modifying the Desktop and Navigation

- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following creates a breadcrumb list of links to the nodes in the selected path.

```
<nav:iterateNavNodesInSelectedPath>
  <nav:navNodeAnchor method="byURL"/>
  <nav:ifHasMoreIterations>

    ... Display separator

  </nav:ifHasMoreIterations>
</nav:iterateNavNodesInSelectedPath>
```

Example

- [iterateSelectedNavNodesLevel \[Page 33\]](#)
- [selectedNavNode \[Page 33\]](#)

3.7.1.2.1.3.19 iterateSelectedNavNodesLevel

Iterates through all the sibling nodes of the node in the selected path on a specified level.

For example, if the user selects *Content Administration* → *Users* → *Create User*, and the tag specifies level 2, then the tag iterates through all the nodes on the same level as the `Users` node (that is, the `Users` node and its siblings).

During each iteration of the tag, a different node is exposed in the body of the tag, either via a cooperating tag that accesses the current node in the iteration or the scriptlet variable defined by the `currentNavNode` attribute.

For more information about the selected path, see [Glossary \[Page 33\]](#).

Attributes

Name	Mandatory	Description
<code>currentNavNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the current node in

Modifying the Desktop and Navigation

		the iteration. The variable is accessible in scriptlets in the body of the tag.
direction	No	Indicates in what direction to iterate. The attribute can have the following values: <ul style="list-style-type: none"> forward (default): Iterate from the first to the last node. backward: Iterate from the last to the first node.
level	Yes	The level in the navigation tree whose nodes to include in the iteration. Only the children of the selected node on the level above are included.

Variables

Name	Scope	Description
<code><currentNavNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the current node in the iteration.

Example

The following creates a detailed navigation iView by starting an iteration of all the sibling nodes of the node in the selected path on level `strStartLevel`.

```
<TABLE Width='100%' Class="lightDTNTable">
  <nav:iterateSelectedNavNodesLevel level="<%=strStartLevel%>"
    currentNavNode="currentRoot">
    <nav:recurseNavNodeChildren currentDepth="currentDepth"
      currentNavNode="currentNavNode">

      ... Display DTN nodes

    </nav:recurseNavNodeChildren>
  </nav:iterateSelectedNavNodesLevel>
</TABLE>
```

See Also

- [iterateNavNodesInSelectedPath \[Page 33\]](#)
- [selectedNavNode \[Page 33\]](#)

3.7.1.2.1.3.20 launchedNavNode

Exposes the navigation node that was launched by the most recent navigation.

The node generally is the same as the node exposed by the `selectedNavNode` tag, but may be different. For example, if the user selects a folder node, the portal launches within the content area the first child node within the folder. The folder node is exposed by the `selectedNavNode` tag and the launched iView or page node is exposed by the `launchedNavNode` tag.

Modifying the Desktop and Navigation

Attributes

Name	Mandatory	Description
navNode	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the node that was launched by the most recent navigation. The variable is accessible in scriptlets in the body of the tag.

Variables

Name	Scope	Description
<code><navNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the launched node.

Example

The following displays the title of the current node.

```
<nav:launchedNavNode navNode="myNode">
```

```
    You are in: <%=myNode.getTitle(request.getLocale())%>
```

```
</nav:launchedNavNode>
```

See Also

- [selectedNavNode \[Page 33\]](#)

3.7.1.2.1.3.21 navNode

Exposes the navigation node with the name specified in the `name` attribute.



The `navNode` tag does not function properly for merged nodes. For more information about merging nodes, see *Merging Navigation Nodes and Defining the Sequence* in the *Portal Administration Guide*.

Attributes

Name	Mandatory	Description
navTarget	Yes	Specifies the name of the navigation node to expose. The name is the same string as returned by the <code>INavigationNode.getName()</code> method. For example, specify <code>ROLES://portal_content/myFolder/myRole</code> to get the node that represents the role <code>myRole</code> in folder <code>portal_content/myFolder</code> .
navNode	No	The name of the Java variable to create for holding the

Modifying the Desktop and Navigation

		INavigationNode object for the specified node. The variable is accessible in scriptlets in the body of the tag.
--	--	--

Variables

<navNode> attribute	No	INavigationNode object representing the specified node.
---------------------	----	---

Example

The following creates a link for a set of navigation nodes, whose names are stored in a `String` array.

```
<% String[] targets = ... %>

...

<nav:navNode navTarget="<%=targets[6]%>">
    <nav:navNodeAnchor method="byURL"/>
</nav:navNode>
```

3.7.1.2.1.3.22 navNodeAnchor

Outputs an HTML link – anchor tag (<a>) with nested text or an image – for navigating to the current navigation node.

For more information about how to create navigation links, see [Triggering Navigation \[Page 33\]](#).

Attributes

Name	Mandatory	Description
anchorAttributes	No	A string that is added inside the anchor tag, in which you can add additional HTML attributes for the anchor tag, such as <code>onClick</code> .
EPCMHistoryMode	No	Indicates whether the navigation node is added to the portal's history list on the page title bar and, if it is, whether duplicate entries are allowed if a user clicks on the link multiple times. The following are valid values: <ul style="list-style-type: none"> • 0: Added to history with duplicates • 1: Added to history without duplicates (default) • 2: No tracking This attribute is only relevant when the <code>navigationMethod</code> attribute is set to <code>byEPCM</code> .

Modifying the Desktop and Navigation

hashURL	No	<p>Indicates whether to set the value of the anchor's <code>href</code> attribute to the standard navigation URL or to its hashed URL.</p> <p>The following are valid values:</p> <ul style="list-style-type: none"> • true (default) • false <p>Any other value is equivalent to <code>true</code>.</p>
navigationContent	No	<p>The navigation node to set as selected.</p> <p>With this attribute, you can launch one navigation node but set another navigation node to appear in navigation iViews as if it was selected.</p> <p>Specify a node by its name, which is the same as the value returned by <code>INavigationNode.getName()</code>, for example, <code>ROLES://portal_content/myFolder/myRole/myIView</code>.</p>
navigationMethod	Yes	<p>Indicates whether to use client-side eventing for navigation after the user clicks the link.</p> <p>The following are valid values:</p> <ul style="list-style-type: none"> • byEPCM: Use client-side eventing. • byURL: Do not use client-side eventing. This eliminates the need for EPCM Javascript, reducing network traffic. This option is useful when implementing an external-facing portal in low-bandwidth scenarios. <p>This option prevents the navigation node from being added to the history list.</p> <p>For more information on an external-facing portal, see Implementing an External-Facing Portal [External].</p>
navigationMode	No	<p>Indicates how to launch the node when the user clicks the link.</p> <p>The following are valid values:</p> <ul style="list-style-type: none"> • 0: Node is launched in the same frame. • 1: Node is launched in new window. • 2: Node is launched in a new session of the portal, which is displayed in a new window. <p>The values are defined by the constants defined in <code>INavigationConstants</code> and with the prefix <code>SHOW</code>.</p>
title	No	<p>The visible text for the link. If not specified, the title of the navigation node is used, which is the same as the value returned by <code>INavigationNode.getTitle()</code>.</p>
urlParameters	No	<p>A string that is appended to the URL in the anchor tag's <code>href</code> attribute. This attribute enables you to add</p>

Modifying the Desktop and Navigation

		parameters to the link URL.
--	--	-----------------------------

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following creates a navigation link.

```
<nav:ifNavNodeSelected>
  <TD nowrap class="chosenTDLevel1">
    <nav:navNodeAnchor navigationMethod="byURL"
      anchorAttributes="class='chosenOnLevel1'"/>
  </TD>
  <% if(levels==2) { %>
    <nav:storeNavNode/>
  <% } %>
</nav:ifNavNodeSelected>
```

3.7.1.2.1.3.23 `navNodeDescription`

Outputs the description for the current navigation node. The string is the same as that returned by `INavigationNode.getDescription()`.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`

Modifying the Desktop and Navigation

- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

3.7.1.2.1.3.24 `navNodeParent`

Exposes the parent node of the current node.

If the current node is a top-level node, the body is not included.

Attributes

Name	Mandatory	Description
<code>navNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the parent node of the current node. The variable is accessible in scriptlets in the body of the tag.

Variables

Name	Scope	Description
<code><navNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the parent node of the node exposed by the cooperating tag.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following displays the title of all siblings of the launched node.

```
<nav:launchedNavNode>
  <nav:navNodeParent>
    <nav:iterateNavNodeChildren>
      <nav:navNodeTitle/>
```

Modifying the Desktop and Navigation

```

        </nav:iterateNavNodeChildren>
    </nav:navNodeParent>
</nav:launchedNavNode>

```

3.7.1.2.1.3.25 navNodePictogram

Outputs an HTML image tag () that displays the image associated with the current node.

If no pictogram is associated with the node, the default pictogram is used.

Attributes

Name	Mandatory	Description
pictogramAttributes	No	A string that is added inside to the image tag, in which you can add additional attributes for the image tag.

Cooperating Tags

The tag must be nested in one of the following tags:

- iterateInitialNavNodes
- iterateNavNodeChildren
- iterateNavNodesInSelectedPath
- iterateSelectedNavNodesLevel
- launchedNavNode
- navNode
- navNodeParent
- recallNavNode
- recurseNavNodeChildren
- selectedNavNode

3.7.1.2.1.3.26 navNodeTitle

Outputs the title for the current navigation node. The string is the same as that returned by `INavigationNode.getTitle()`.

Cooperating Tags

The tag must be nested in one of the following tags:

- iterateInitialNavNodes
- iterateNavNodeChildren
- iterateNavNodesInSelectedPath
- iterateSelectedNavNodesLevel
- launchedNavNode
- navNode
- navNodeParent

Modifying the Desktop and Navigation

- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

3.7.1.2.1.3.27 `recallNavNode`

Exposes a navigation node that was previously stored by the `storeNavNode` tag.

Attributes

Name	Mandatory	Description
<code>id</code>	No	The ID of the stored node. If no ID is specified, the last node that was stored with no ID or with the ID <code>default</code> is recalled.

Example

The following iterates through the top-level nodes, storing the selected top-level node. The selected node is then recalled, and an iteration is started through the children of that node.

```
<TABLE border="0" cellspacing="0" cellpadding="0" class="table1">
  <TR>
    <nav:iterateInitialNavNodes>
      <nav:ifNavNodeSelected>

        ...

        <% if(levels==2) { %>
          <nav:storeNavNode/>
        <% } %>
      </nav:ifNavNodeSelected>
      <nav:ifNotNavNodeSelected>

        ...

      </nav:ifNotNavNodeSelected>
    </nav:iterateInitialNavNodes>
    <TD nowrap class="spacingTDLevel1">&nbsp;  </TD>
  </TR>
</TABLE>
<TABLE border="0" cellspacing="0" cellpadding="0" class="table2">
  <TR>
    <nav:recallNavNode>
      <nav:iterateNavNodeChildren>

        ... Display level 2 nodes

      </nav:iterateNavNodeChildren>
      <TD nowrap class="spacingTDLevel2">&nbsp;  </TD>
    </nav:recallNavNode>
  </TR>
</TABLE>
```

See Also

- [storeNavNode \[Page 33\]](#)

3.7.1.2.1.3.28 recurseNavNodeChildren

Iterates recursively through all the nodes below the current node, performing a depth-first traversal through the navigation tree, starting with the current node.

During each iteration of the tag, a different node is exposed in the body of the tag, either via a cooperating tag that accesses the current node in the iteration or the scriptlet variable defined by the `currentNavNode` attribute.

Attributes

Name	Mandatory	Description
<code>depthLimit</code>	No	The number of levels through which to iterate.
<code>currentDepth</code>	No	The name of the Java variable to create for holding the <code>Byte</code> object that equals the current depth within the recursive iteration. The first level of the iteration equals 1. The variable is accessible in scriptlets in the body of the tag.
<code>currentNavNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the current node in the iteration. The variable is accessible in scriptlets in the body of the tag.

Variables

Name	Scope	Description
<code><currentNavNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the current node in the iteration.
<code><currentDepth></code> attribute	Body of tag	A <code>Byte</code> object representing the current depth within the recursive iteration.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`

Modifying the Desktop and Navigation

- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following creates a detailed navigation iView by starting a recursive iteration of all the children nodes of the node in the selected path on the specified level. The current depth is used in the `TR` tag to create the required indent for that level.

```
<TABLE Width='100%' Class="lightDTNTable">
  <nav:iterateSelectedNavNodesLevel level="<%=strStartLevel%>"
    currentNavNode="currentRoot">
    <nav:recurseNavNodeChildren currentDepth="currentDepth"
      currentNavNode="currentNavNode">

      ...

      <TR Style="{text-indent:<%=((currentDepth.byteValue()-1) *
16)+10%>px}">
        <TD Class="lightDTNCell">

          ...

        </TD>
      </TR>
    </nav:recurseNavNodeChildren>
  </nav:iterateSelectedNavNodesLevel>
</TABLE>
```

Example

- [doNotRecurseNavNodeChildren \[Page 33\]](#)

3.7.1.2.1.3.29 selectedNavNode

Exposes the navigation node that was selected by the user during the most recent navigation.

Attributes

Name	Mandatory	Description
<code>navNode</code>	No	The name of the Java variable to create for holding the <code>INavigationNode</code> object for the node that was selected by the user. The variable is accessible in scriptlets in the body of the tag.

Variables

Name	Scope	Description
<code><navNode></code> attribute	Body of tag	<code>INavigationNode</code> object representing the selected node.

Modifying the Desktop and Navigation

See Also

- [launchedNavNode \[Page 33\]](#)
- [ifNavNodeEqualsSelectedNavNode \[Page 33\]](#)
- [ifNavNodeInSelectedPath \[Page 33\]](#)

3.7.1.2.1.3.30 storeNavNode

Stores a navigation node so that it can later be recalled by the `recallNavNode` tag.

Attributes

Name	Mandatory	Description
id	No	The ID of the stored navigation node. If no ID is specified, the navigation node is stored with the ID default.

Cooperating Tags

The tag must be nested in one of the following tags:

- `iterateInitialNavNodes`
- `iterateNavNodeChildren`
- `iterateNavNodesInSelectedPath`
- `iterateSelectedNavNodesLevel`
- `launchedNavNode`
- `navNode`
- `navNodeParent`
- `recallNavNode`
- `recurseNavNodeChildren`
- `selectedNavNode`

Example

The following iterates through the top-level nodes, storing the selected top-level node. The selected node is then recalled, and an iteration is started through the children of that node.

```
<TABLE border="0" cellspacing="0" cellpadding="0" class="table1">
  <TR>
    <nav:iterateInitialNavNodes>
      <nav:ifNavNodeSelected>
        ...
        <% if(levels==2) { %>
          <nav:storeNavNode/>
        <% } %>
      </nav:ifNavNodeSelected>
    <nav:ifNotNavNodeSelected>
```

Modifying the Desktop and Navigation

```

        ...
    </nav:ifNotNavNodeSelected>
</nav:iterateInitialNavNodes>
    <TD nowrap class="spacingTDLevel1">&nbsp;</TD>
</TR>
</TABLE>
<TABLE border="0" cellspacing="0" cellpadding="0" class="table2">
    <TR>
        <nav:recallNavNode>
            <nav:iterateNavNodeChildren>

                ... Display level 2 nodes

            </nav:iterateNavNodeChildren>
            <TD nowrap class="spacingTDLevel2">&nbsp;</TD>
        </nav:recallNavNode>
    </TR>
</TABLE>

```

See Also

- [recallNavNode \[Page 33\]](#)

3.7.1.2.1.4 Samples

This section provides samples of the following:

- [Top-Level Navigation iView \[Page 33\]](#)
- [Detailed Navigation iView \[Page 33\]](#)

3.7.1.2.1.4.1 Top-Level Navigation iView

The following creates a one- or two-level top-level navigation iView, depending on the value in the `NumOfDisplayLevels` property of the navigation iView portal component. The page does the following:

- Iterates through the initial nodes.
- Saves the node that is selected.
- After iterating through all the initial nodes, recalls the selected node and iterates through its children.

```

<%@ page import = "com.sapportals.portal.navigation.*" %>
<%@ page import = "com.sapportals.portal.prt.resource.IResource" %>
<%@ page import = "com.sapportals.portal.prt.runtime.PortalRuntime"
%>
<%@ page import =
"com.sapportals.portal.navigation.INavigationGenerator"%>
<%@ page import =
"com.sapportals.portal.prt.component.IPortalComponentRequest" %>

<%@ taglib uri="NavigationTagLibrary" prefix="nav" %>

<%
INavigationGenerator service =

```


Modifying the Desktop and Navigation

```

(INavigationGenerator) PortalRuntime.getRuntimeResources().getService(
    INavigationService.KEY);

IPortalComponentRequest currentRequest =
    (IPortalComponentRequest) pageContext.getAttribute(
        javax.servlet.jsp.PageContext.REQUEST);

IPortalComponentResponse currentResponse =
    (IPortalComponentResponse) pageContext.getAttribute(
        javax.servlet.jsp.PageContext.RESPONSE);

IPortalComponentContext componentContext =
    currentRequest.getComponentContext();

IPortalComponentProfile profile = componentContext.getProfile();

boolean previewMode = false;
String mode = (String) componentRequest.getNode().getValue("mode");
if(mode != null && mode.equals("preview")) {
    previewMode = true;
}

String strLevels = profile.getProperty("NumOfDisplayLevels");
int levels = 2;
try {
    levels = Integer.parseInt(strLevels);
} catch(NumberFormatException e) {
}
if(levels < 0 || levels > 2) {
    levels = 2;
}
if(levels > 0) { %>

<TABLE border="0" cellspacing="0" cellpadding="0" class="mainTable">
  <TR>
    <TD nowrap class="notch">&nbsp;</TD>
    <TD>
      <TABLE border="0" cellspacing="0" cellpadding="0"
class="table1">
        <TR>

          <!-- Iterate initial nodes -->
          <nav:iterateInitialNavNodes>
            <nav:ifNavNodeInSelectedPath>
              <TD nowrap class="chosenTDLevel1">
                <% if(previewMode) { %>
                  <A class="chosenOnLevel1"><nav:navNodeTitle/></A>
                <% } else { %>
                  <nav:navNodeAnchor navigationMethod="byURL"
                    anchorAttributes="class='chosenOnLevel1'"/>
                <% } %>
              </TD>
              <% if(levels==2) { %>
                <!-- Save selected node -->
                <nav:storeNavNode/>
              <% } %>
            </nav:ifNavNodeInSelectedPath>
            <nav:ifNotNavNodeInSelectedPath>

```

Modifying the Desktop and Navigation

```

        <TD nowrap class="unChosenTDLevel1">
            <% if(previewMode) { %>
                <A class="chosenOnLevel1"><nav:navNodeTitle/></A>
            <% } else { %>
                <nav:navNodeAnchor navigationMethod="byURL"
                    anchorAttributes="class='unChosenOnLevel1'"/>
            <% } %>
        </TD>
    </nav:ifNotNavNodeInSelectedPath>
</nav:iterateInitialNavNodes>
    <TD nowrap class="spacingTDLevel1">&nbsp;</TD>
</TR>
</TABLE>
<TABLE border="0" cellspacing="0" cellpadding="0"
class="table2">
    <TR>

        <!--Recall initial selected node and iterate
children -->
        <nav:recallNavNode>
            <nav:iterateNavNodeChildren>
                <nav:ifNavNodeInSelectedPath>
                    <TD nowrap class="chosenTDLevel2">
                        <% if(previewMode) { %>
                            <A class="chosenOnLevel2"><nav:navNodeTitle/></A>
                        <% } else { %>
                            <nav:navNodeAnchor navigationMethod="byURL"
                                anchorAttributes="class='chosenOnLevel2'"/>
                        <% } %>
                    </TD>
                </nav:ifNavNodeInSelectedPath>
                <nav:ifNotNavNodeInSelectedPath>
                    <TD nowrap class="unChosenTDLevel2">
                        <% if(previewMode) { %>
                            <A
class="unChosenOnLevel2"><nav:navNodeTitle/></A>
                        <% } else { %>
                            <nav:navNodeAnchor navigationMethod="byURL"
                                anchorAttributes="class='unChosenOnLevel2'"/>
                        <% } %>
                    </TD>
                </nav:ifNotNavNodeInSelectedPath>
                <nav:ifHasMoreIterations>
                    <TD nowrap class="spacingTDPipeLevel2"> | </TD>
                </nav:ifHasMoreIterations>
            </nav:iterateNavNodeChildren>
            <TD nowrap class="spacingTDLevel2">&nbsp;</TD>
        </nav:recallNavNode>
    </TR>
</TABLE>
</TD>
</TR>
</TABLE>
<% } %>

```

Modifying the Desktop and Navigation

3.7.1.2.1.4.2 Detailed Navigation iView

The following creates a detailed navigation iView. The page does the following:

- Determines the start level for the detailed navigation.
- Starts an iteration through the children nodes of the selected node on the level above the start level.
- For each node, starts a recursive iteration through all its children nodes.
- For each node in the recursive iteration, displays the appropriate HTML.

```
<%@ page import = "java.util.*" %>
<%@ page import = "com.sapportals.portal.navigation.*" %>
<%@ page import = "com.sapportals.portal.prt.resource.IResource" %>
<%@ page import = "com.sapportals.portal.prt.runtime.PortalRuntime"
%>
<%@ page import =
"com.sapportals.portal.navigation.INavigationGenerator"%>
<%@ page import =
"com.sapportals.portal.prt.component.IPortalComponentRequest" %>

<%@ taglib uri="NavigationTagLibrary" prefix="nav" %>

<%
NavigationEventsHelperService helperService =

(NavigationEventsHelperService) PortalRuntime.getRuntimeResources()
    .getService(NavigationEventsHelperService.KEY);

INavigationGenerator service =
    (INavigationGenerator) PortalRuntime.getRuntimeResources()
        .getService(INavigationService.KEY);

IPortalComponentRequest currentRequest =
    (IPortalComponentRequest) pageContext.getAttribute(
        javax.servlet.jsp.PageContext.REQUEST );

IPortalComponentResponse currentResponse =
    (IPortalComponentResponse) pageContext.getAttribute(
        javax.servlet.jsp.PageContext.RESPONSE);

IPortalComponentContext componentContext =
    currentRequest.getComponentContext();

IPortalComponentProfile profile = componentContext.getProfile();

IPortalComponentURI componentURI =
    currentRequest.createPortalComponentURI ();

TreeSet openNodes = (TreeSet)currentRequest
    .getComponentSession().getValue("LightDTNOpenNodes");

if(openNodes==null) {
    openNodes = new TreeSet();
    session.putValue("LightDTNOpenNodes", openNodes);
}
```

Modifying the Desktop and Navigation

```

String oldIndices = (String) currentRequest
    .getComponentSession().getValue("LightDTNRootIndicies");

NavigationNodes path = (NavigationNodes) helperService
    .getNavNodesListForPath(
        currentRequest, INavigationConstants.NAVIGATION_CONTEXT_ATTR);

boolean nodeIsOpen          = false;
boolean nodeIsPressed       = false;
boolean nodeInSelectedPath  = false;
int    nodeID               = 0;
Integer nodeIDInteger       = null;
int    pressedKnobID        = 0;
Integer pressedKnobIDInteger = null;
String pressedKnobIDString = (String) currentRequest.
    getParameter("LightDTNKnobID");
String urlParameters        = "";

boolean previewMode = false;
String mode = (String) componentRequest.getNode().getValue("mode");

if(mode!=null && mode.equals("preview")) {
    previewMode = true;
}

componentURI.setContextName(currentRequest.getComponentContext().
    getContextName());

componentURI.setTargetNode(currentRequest.getNode());

if(pressedKnobIDString!=null && !pressedKnobIDString.equals("")) {
    pressedKnobID        = Integer.parseInt(pressedKnobIDString);
    pressedKnobIDInteger = new Integer(pressedKnobID);
}

// Find start level
String strStartLevel = profile.getProperty(
    "com.sap.portal.navigation.lightdetailednavigationtree.startLevel");
int startLevel = 3;
try {
    startLevel = Integer.parseInt(strStartLevel);
} catch(NumberFormatException e) {
    strStartLevel = "3";
}
if(startLevel<1 || previewMode) {
    startLevel = 1;
    strStartLevel = "1";
}

String    indices        = "";
ArrayList indicesList =
    helperService.getNavNodesPathIndexesList(currentRequest);
try {
    for(int i=0; i<startLevel-1; i++) {
        indices += indicesList.get(i) + ".";
    }
}

```

Modifying the Desktop and Navigation

```

} catch(IndexOutOfBoundsException e) {
}
if(oldIndices==null || !indices.equals(oldIndices)) {
    openNodes.clear();
}
if(!indices.equals("")) {
    currentRequest.getComponentSession().putValue(
        "LightDTNRootIndicies",indices);
}
%>

<TABLE Width='100%' Class="lightDTNTable">

    <!--Iterate through children of selected node in TLN -->
    <nav:iterateSelectedNavNodesLevel level="<%=strStartLevel%>">

        <!--For each node, start recursive iteration -->
        <nav:recurseNavNodeChildren currentDepth="currentDepth"
            currentNavNode="currentNavNode">
            <% nodeInSelectedPath = false; %>
            <nav:ifNavNodeInSelectedPath>
                <% nodeInSelectedPath = true; %>
            </nav:ifNavNodeInSelectedPath>
            <% nodeID
                = currentNavNode.getName().hashCode();
                nodeIDInteger = new Integer(nodeID);
                nodeIsOpen    = openNodes.contains(nodeIDInteger);
                nodeIsPressed = (pressedKnobIDInteger!=null &&
                                pressedKnobID==nodeID);
                urlParameters = "LightDTNKnobID=" + nodeID;
                if(nodeIsOpen) {
                    if(nodeIsPressed) {
                        openNodes.remove(nodeIDInteger);
                        nodeIsOpen = false;
                    }
                } else {
                    if(nodeIsPressed) {
                        openNodes.add(nodeIDInteger);
                        nodeIsOpen = true;
                    }
                }
                if(nodeInSelectedPath && pressedKnobIDInteger==null) {
                    openNodes.add(nodeIDInteger);
                    nodeIsOpen = true;
                }
            %>

            <!--Display HTML for each node -->
            <TR Style="{display:block;position:relative;left:
                <%=((currentDepth.byteValue()-1) * 16)+10%>px}">
            <TD Class="lightDTNCell">
                <nav:ifNavNodeIsFolder>
                    <nav:ifNavNodeHasChildren>
                        <% if(previewMode) { %>
                            <A class="lightDTNKnob<%
                                if(nodeIsOpen) { %>Open<% } else { %>Closed<% }
                                %>" style="display:block;float:left"/>
                            <% } else { %>

```

Modifying the Desktop and Navigation

```

        <A href="<%=componentURI%>?<%=urlParameters%>"
            class="lightDTNKnob<%
                if(nodeIsOpen) { %>Open<% } else { %>Closed<% }
            %>" style="display:block;float:left"/>
    <% }
        if(!nodeIsOpen) { %>
            <nav:doNotRecurseNavNodeChildren/>
            <% } %>
        </nav:ifNavNodeHasChildren>
        <nav:ifNotNavNodeHasChildren>
            <A class="lightDTNKnobNone"
style="display:block;float:left"/>
        </nav:ifNotNavNodeHasChildren>
        <% if (previewMode) { %>
            <A class="lightDTN<%
                if(nodeIsOpen) { %>Open<% } else { %>Closed<% }
            %>Folder" style="display:block;float:left"/>
        <% } else { %>
            <A href="<%=componentURI%>?<%=urlParameters%>"
                class="lightDTN<%
                    if(nodeIsOpen) { %>Open<% } else { %>Closed<% }
                %>Folder" style="display:block;float:left"/>
            <% } %>
        </nav:ifNavNodeIsFolder>
        <nav:ifNotNavNodeIsFolder>
            <nav:doNotRecurseNavNodeChildren/>
            <A class="lightDTNKnobNone"
style="display:block;float:left"/>
            <A class="lightDTNLeaf"
style="display:block;float:left"/>
        </nav:ifNotNavNodeIsFolder>
        <nav:ifNavNodeEqualsLaunchedNavNode>
            <% if (previewMode) { %>
                <A class="lightDTNTextSelected"
                    style="float:left"><nav:navNodeTitle/></A>
            <% } else { %>
                <nav:navNodeAnchor navigationMethod="byURL"
                    urlParameters="<%=urlParameters%>"
                    anchorAttributes="class='lightDTNTextSelected'
                    style='float:left'"/>
            <% } %>
        </nav:ifNavNodeEqualsLaunchedNavNode>
        <nav:ifNotNavNodeEqualsLaunchedNavNode>
            <nav:ifNavNodeIsFolder>
                <% if (nodeIsOpen) { %>
                    <A href="<%=componentURI%>?<%=urlParameters%>"
                        class="lightDTNText"
                        style='float:left'><nav:navNodeTitle/></A>
                <% } else {
                    if (previewMode) { %>
                        <A class="lightDTNText"><nav:navNodeTitle/></A>
                    <% } else { %>
                        <nav:navNodeAnchor navigationMethod="byURL"
                            urlParameters="<%=urlParameters%>"
                            anchorAttributes="class='lightDTNText'
                            style='float:left'"/>
                    <% }
                <% }
            </nav:ifNavNodeIsFolder>
        </nav:ifNotNavNodeEqualsLaunchedNavNode>
    <% } %>

```

Modifying the Desktop and Navigation

```

        </nav:ifNavNodeIsFolder>
        <nav:ifNotNavNodeIsFolder>
            <% if (previewMode) { %>
                <A class="lightDTNText"
                    style='float:left'><nav:navNodeTitle/></A>
            <% } else { %>
                <nav:navNodeAnchor navigationMethod="byURL"
                    anchorAttributes="class='lightDTNText'
style='float:left'"/>
            <% } %>
        </nav:ifNotNavNodeIsFolder>
        </nav:ifNotNavNodeEqualsLaunchedNavNode>
    </TD>
</TR>
</nav:recurseNavNodeChildren>
</nav:iterateSelectedNavNodesLevel>
</TABLE>

```

3.7.1.2.2 Framework Tag Library

The framework tag library helps you create masthead and page title bar navigation iViews. The tags enable you to create standard links (such as for logging off) and display commonly used text strings (such as the current user's name).

For all tags that create a link, the tag's body is the text or image for the link.

None of the tags has attributes.

Tags for Masthead iViews

The masthead iView is generally used to display the current user's name, links for logging off the portal and performing other tasks, and branding images and text.

The following tags are available:

Tag	Description
personalizePortalAnchor	Creates a link to open the personalize dialog for the portal.
newSessionAnchor	Creates a link to open a new session in the portal.
loginAnchor	Creates a link to log on to the portal.
logoutAnchor	Creates a link to log off the portal.
portalHelpAnchor	Creates a link to display help for the portal.
userFirstName	Displays the current user's first name.
userLastName	Displays the current user's last name.
welcomeTitle	Displays the default welcome message.



The masthead tags require special JavaScript contained in the default masthead. To create your own masthead iView, either modify a copy of the

Modifying the Desktop and Navigation

default masthead or copy the JavaScript from the default masthead to your masthead.

Tags for Page Title Bar iViews

The page title bar iView is generally used to display the title of the current page and to display links for invoking page functions, such as refreshing or personalizing the page.

The following tags are available:

Tag	Description
pageDetailsAnchor	Creates a link to display information about the current page.
pageExpandAnchor	Creates a link to open the current page in a new window.
pageHelpAnchor	Creates a link to display help for the current page.
pagePersonalizeAnchor	Creates a link to open the personalize dialog for the current page.
pageRefreshAnchor	Creates a link to refresh the page.
addToBrowserFavoritesAnchor	Creates a link to add the current page to the browser's favorites.
addToPortalFavoritesAnchor	Creates a link to add the current page to the portal favorites displayed in the navigation panel.

3.7.1.2.2.1 How to Use the Tag Library

The framework tag library is designed to enable you to build navigation iViews based on JSP pages, which are deployed to the portal as portal components in a PAR file that contains the JSP page and a `portalapp.xml` file.

For more information on writing JSP pages for the portal, see [Writing JSP Pages \[Page 33\]](#).

Referencing the Tag Library

To use navigation tags in a JSP page, do the following:

1. At the top of the JSP page, add a reference to the tag library, as follows:

```
<%@ taglib uri="FrameworkTagLibrary" prefix="nav" %>
```

2. In the `portalapp.xml` file:

- a. Add a sharing reference in the `<application-config>` element, as follows:

```
<property name="SharingReference"
  value="com.sap.portal.pagebuilder"/>
```

- b. Add a reference to the tag library by adding the following property to the `<component-profile>` element for the JSP component:

```
<property name="FrameworkTagLibrary"
  value="/SERVICE/com.sap.portal.pagebuilder/taglib/framework.tld"/>
```


3.7.1.3 Creating Navigation Connectors

Purpose

The navigation service creates a tree of navigation nodes for the current user from the registered navigation connectors. Navigation iViews can display these nodes to enable the user to navigate within the portal.

The portal comes with the roles navigation connector, which creates nodes based on objects defined in the PCD and is closely linked with the roles hierarchy defined by a content administrator. A user administrator assigns roles to individual users. The roles navigation connector then creates a set of navigation nodes based on the current user and the roles assigned to the user.

You can create additional connectors for creating additional navigation nodes.

A navigation connector is packaged in a portal application (PAR file). The following are the steps for creating a navigation connector:

- [Step 1: Creating a Navigation Connector Node \[Page 33\]](#)
- [Step 2: Creating a Navigation Connector \[Page 33\]](#)
- [Step 3: Registering the Connector \[Page 33\]](#)

Several methods of your navigation connector (`INavigationConnector`) and navigation connector node (`INavigationConnectorNode`) return a `javax.naming.NamingEnumeration` object. You may need to write an implementation for this interface. The examples in this section make use of the class `NavigationEnum`, which is a sample implementation of `NamingEnumeration` for this example only. The implementation is not described here.

3.7.1.3.1 Step 1: Creating a Navigation Connector Node

Your navigation connector is, essentially, a supplier of navigation nodes. Each navigation node is an `INavigationConnectorNode` object, and you must create a class that determines the values of navigation attributes.

Procedure

1. Create a new class that extends `AbstractNavigationConnectorNode`.
2. Implement the `INavigationConnectorNode` methods that return attribute values for the navigation connector node – methods that start with `get` and `is`.
3. Implement `listBindings()`, which returns a `javax.naming.NamingEnumeration` of nodes related to the current node, for example, the children of the current node. The mode parameter determines the type of nodes to return, which can be one of the following:
 - Children of the current node
 - First child of the current node
 - Dynamic navigation iViews for the current node
 - Drag&Relate targets for the current node
 - Related links for the current node

Modifying the Desktop and Navigation

The following is a sample implementation, in which the navigation connector node class holds the relevant nodes in several local variables and returns the relevant set of nodes depending on the mode. For example, `childrenNodeBindings` is a `List` that holds the children of the current node.

The mode constants are defined by the `INavigationConnectorNode` interface.

```
public NamingEnumeration listBindings(String binding, String mode,
    Hashtable filterParameters)
    throws NamingException {

    if (mode.equals(NAVIGATION_GET_CHILDREN))
        return new NavigationEnum(childrenNodeBindings);
    if (mode.equals(NAVIGATION_GET_RELATED_SEE_ALSO))
        return new NavigationEnum(relatedSeeAlsoNodeBindings);
    if (mode.equals(NAVIGATION_GET_RELATED_DR_TARGETS))
        return new NavigationEnum(relatedTargetNodeBindings);
    if (mode.equals(NAVIGATION_GET_FIRST_CHILD)) {
        return new NavigationEnum(
            Collections.singletonList(childrenNodeBindings.get(0)));
    }
    else throw new NamingException("Unknown mode named "+mode);
}
```

The following values are passed into `listBinding()` via the `Hashtable` parameter:

- User (`NavigationPrincipal` key)
- Navigation target (`NavigationTarget` key)

3.7.1.3.2 Step 2: Creating a Navigation Connector

The navigation connector implements methods that define the following:

- The initial (top-level) nodes of the connector. Once the navigation connector returns the initial nodes, the navigation service can query the nodes themselves for each one's children and related nodes.
- The mapping between a navigation URL and the nodes defined by the connector.

For example, if the connector prefix is `myPrefix`, a navigation URL for the connector could be `myPrefix://myParentNode2/myChildNode5`. The connector must determine what `INavigationConnectorNode` object to return for this URL.

Procedure

1. Create a new class that extends `AbstractNavigationConnector`.
2. Implement the method `getInitialNodes()`, which returns a `javax.naming.NamingEnumeration` of the initial nodes. The method could return the nodes only if a condition is true, for example, the user is part of specific role.

The following returns the initial nodes if the user is part of the Java developer role:

```
public NamingEnumeration getInitialNodes(Hashtable environment) {
    if (isUserInJavaDeveloperRole(environment))
        return new NavigationEnum(initialNodes);
    else
        return NavigationEnum.EMPTY_ENUM;
}
```

Modifying the Desktop and Navigation

3. Implement the method `getNode()`, which returns a `INavigationConnectorNode` based on a navigation URL passed into the method.

You can implement any logic for returning the `INavigationConnectorNode` object. One method is to store all the nodes in a `Map` with the navigation URL as the key. To implement this, you could do the following:

- a. Store the nodes in a `Map`. You can add code for this in an initialization method.

One implementation is to create a method that recursively traverses the navigation nodes that you created and store them in a map with the fully qualified name (navigation URL) as the key for each one, as follows:

```
private void recursivelyFillNamesToNodesMap(NamingEnumeration enum)
    throws NamingException {
    while (enum.hasMoreElements()) {
        Binding b = (Binding) enum.nextElement();
        myConnectorNode node = (myConnectorNode) b.getObject();
        String nodeName = node.getName();
        namesToNodes.put(nodeName, node);
        atomicNamesToNames.put(node.getAtomicName(), nodeName);
        recursivelyFillNamesToNodesMap(node.listBindings(
            "", INavigationConnectorNode.NAVIGATION_GET_CHILDREN));
    }
}
```

You could then call `recursivelyFillNamesToNodesMap` from the initialization method, passing in the initial nodes.

- b. Look up the node in the `Map` based on the navigation URL. The following returns a node in the `namesToNodes` `Map` if the user is a member of the Java developer role:

```
public INavigationConnectorNode getNode(
    Hashtable environment, String connectorNodeURL) {
    boolean memberOfRole = isUserInJavaDeveloperRole(environment);
    if (memberOfRole)
        return (myConnectorNode) namesToNodes.get(connectorNodeURL);
    else
        return null;
}
```

4. Implement the method `getNodes()`, which returns a `javax.naming.NamingEnumeration` of `INavigationConnectorNode` objects based on a set of navigation URLs passed into the method, as follows:

```
public NamingEnumeration getNodes(
    Hashtable environment, Vector connectorNodeURLs) {
    myConnectorNode node;

    int size = connectorNodeURLs.size();
    List nodeBindings = new ArrayList(size);

    for (int i = 0; i < size; i++) {
        node = (myConnectorNode) getNode(
            environment, (String)
connectorNodeURLs.get(i));
        if (node != null) nodeBindings.add(
            new Binding(node.getName(), node));
    }

    return new NavigationEnum(nodeBindings);
}
```

Modifying the Desktop and Navigation

```
}

```

5. Implement the method `getNodeByQuickLink()` in order to map quick links strings to specific navigation nodes.

For more information, see [Quick Links \[Page 33\]](#).

6. Implement `getConnectorCacheDiscriminator()`, which assigns a unique key to each unique set of initial nodes. This enables the portal to cache navigation nodes, improving performance.

7. By default, nodes from the navigation connector are cacheable. Override `isCacheable()` to indicate whether the nodes from the navigation connector are cacheable.

For more information, see [Navigation Cache \[Page 33\]](#).

3.7.1.3.3 Step 3: Registering the Connector

You must register the connector with the navigation service for the navigation service to include your navigation nodes in the portal.

Generally, you create a new service whose only function is to register your connector.

Procedure

1. Create a portal service, which must implement `IService`.
2. Define a constant for the prefix.

```
public static String NAV_CONNECTOR_PREFIX = "myPrefix";

```

3. In the service's `init()` method, create an instance of your `INavigationConnector` class.

```
public void init(IServiceContext serviceContext) {
    mm_serviceContext = serviceContext;
    myConnector = new myConnector();
}

```

4. Register the navigation connector in the service's `afterInit()` method.

```
public void afterInit() {
    INavigationConnectorRegistration service =
        (INavigationConnectorRegistration)
            getContext().getService(INavigationService.KEY);
    if (service != null) {
        service.registerConnector(
            NAV_FILE_CONNECTOR_PREFIX, myConnector);
    }
}

```

5. Create a service entry in `portalapp.xml` for the service, similar to the one shown below:

```
<service name="myConnectorService">
  <service-config>
    <property name="className" value="myConnectorService"/>
    <property name="startup" value="true"/>
  </service-config>
</service>

```

3.7.1.3.4 Redirectors

The portal enables you to create redirectors that automatically translate all navigation targets with a specific prefix into different navigation targets.

For example, the roles connector includes a redirector (that is, a class that implements `INavigationRedirector`) that translates any navigation target with the `pcd` prefix into the same navigation target but with a `ROLES` prefix. This enables users or applications to navigate to role-based navigation nodes by specifying a target with a `pcd` prefix.



You do not have to implement a redirector for your navigation connector. However, you can only register a redirector at the same time you register a navigation connector.

Procedure

The following describes how to write and deploy a redirector:

1. Create a new class that implements `INavigationRedirector`.
2. Implement the method `redirect()`, which has the following signature:

```
public INavigationRedirectorResult redirect(  
    String atomicName, Hashtable hashtable) throws  
    NamingException {  
}
```

The original navigation target is passed as a string, without a prefix or separator.

The method provides the logic for translating this string into the new navigation target, including a prefix and separator. The method returns an object of type `INavigationRedirectorResult`. You must create a class that implements this interface.

3. Package this class with the PAR file that contains your navigation connector.
4. Register the redirector in the same call that registers your navigation connector, as follows
 - a. Create an instance of your redirector.
 - b. Create a `Map` for holding your redirectors.
 - c. Register your redirector when registering your connector.

```
private myConnectorRedirector myConnectorRedirector;  
  
// Create instance of redirector.  
public void init(IServiceContext serviceContext) {  
    myConnectorRedirector = new myConnectorRedirector();  
}
```

Modifying the Desktop and Navigation

```
// Register redirector.
public void afterInit() {

    INavigationConnectorRegistration service =
        (INavigationConnectorRegistration) getContext().
            getService(INavigationService.KEY);
    if (service != null) {
        Map redirectors = new HashMap();
        redirectors.put("myRedirectPrefix", myConnectorRedirector);

        service.registerConnector(
            "myRedirectPrefix", myConnector, null, redirectors);
    }
}
```

3.7.1.4 Triggering Navigation

The following describes the methods for creating links to other navigation nodes:

- **Navigation Tag Library (JSP)**

The navigation tag library provides access to navigation nodes and can create navigation links from them.

For example, the following exposes the navigation node

ROLES://portal_content/myRole/myIView and creates a link for navigating to the node:

```
<nav:navNode navTarget="ROLES://portal_content/myRole/myIView">
  <nav:navNodeAnchor navigationMethod="byURL"
    urlParameters="<%=urlParameters%>"
    anchorAttributes="class='myCSSstyle'"/>
</nav:navNode>
```

The tag library is for portal components built from a JSP page.

For more information, see [Navigation Tag Library \[Page 33\]](#).

- **Client-Side Eventing (EPCM)**

The portal's client-side eventing mechanism (EPCM) provides a JavaScript function for navigating to a specific navigation node.

The following creates a link for navigating to the

ROLES://portal_content/myRole/myIView node:

```
<A HREF="myLink"
  onclick="return EPCM.doNavigate
    ('ROLES://portal_content/myRole/myIView')">
  This is an HTML Link
</A>
```

For more information, see [Client-Side Eventing \[Page 33\]](#)

Instead of writing the code for the EPCM.doNavigate call, you can get the code by calling NavigationEventsHelper.addClickEvent(), which creates the code and stores it as a StringBuffer in one of the function's parameter, as shown in the following:

```
import com.sapportals.htmlb.Link;
import com.sapportals.portal.navigation.NavigationEventsHelper;;
```

Modifying the Desktop and Navigation

```
import
com.sapportals.portal.navigation.NavigationEventsHelperService;

NavigationEventsHelperService service =
(NavigationEventsHelperService)
    PortalRuntime.getRuntimeResources().getService(
        NavigationEventsHelperService.KEY);

NavigationEventsHelper helper =
    service.getNavigationEventsHelperInstance();

INavigationNode currNode = service.getCurrentContextNavNode(request);

Link titleLink = new Link(
    currNode.getName(), currNode.getTitle(request.getLocale()));

StringBuffer scriptTarget = new StringBuffer();

helper.addClickEvent(
    currNode, scriptTarget, request, false, null);

titleLink.setReference("javascript:void(0)");
titleLink.setOnClientClick(scriptTarget.toString());
titleLink.setLinkDesign(LinkDesign.DRAGRELATE);
```

- **Portal Runtime Link (PRT)**

You can create a link to a specific portal component instead of to a node in the navigation hierarchy. Use the PRT API to create a link, as follows:

```
IPortalComponentURI componentURI =
request.createPortalComponentURI();
componentURI.setContextName("myApplication.myComponent");
myUrl = componentURI.toString();
```

Embed the `myURL` string in an HTML anchor tag.

For more information on portal components, see [Portal Runtime \[Page 2\]](#).

3.7.2 Creating Custom Layouts

You can create a custom page layout to display content in a way not possible with the standard layouts provided with the portal.

A layout is defined in a JSP page that is packaged as a component in a PAR file and deployed to the portal. Once deployed, an administrator can create a layout template based on the component, and then create pages based on the layout template.

Prerequisites

- You are familiar with how portal pages are rendered, as described in [Portal Page at Runtime \[External\]](#).
- You are familiar with how to write JSP pages in the portal, as described in [Writing JSP Pages \[Page 33\]](#).

Modifying the Desktop and Navigation

3.7.2.1 How to Create a Custom Layout

A layout consists of a layout JSP page and a `portalapp.xml` file. The files are packaged in a portal application (PAR file) and deployed to the portal.

You can define several layouts in a single PAR by providing for each layout a JSP page and a `<component>` element in the `portalapp.xml` file.

Procedure

- With the help of the layout tag library, create a JSP page that defines the following:
 - One or more containers on the page in which iViews can be placed.
 - The iView tray, or frame, in which each iView is placed. If you do not specify a tray, the portal displays the default tray, which provides links to standard tray functions, such as hiding, refreshing or personalizing the iView.
You can specify one tray for each container. The tray is displayed for all iViews in the container.
 - Additional HTML that appears on each page.

For more information on creating a layout JSP page, see [Layout Tag Library \[Page 33\]](#).

- Create a `<component>` element in a `portalapp.xml` file for the PAR and set the `name` attribute. Create one `<component>` element for each layout defined in the PAR.

In the `<component>` element's `<component-config>` element, create the following `<property>` elements:

Property	Mandatory	Description/Value
<code>ClassName</code>	Yes	<code>com.sapportals.portal.pb.layout.PageLayout</code>
<code>ResourceBundleName</code>	Yes	<code>pagebuilder_nls</code>

In the `<component>` element's `<component-profile>` element, create the following `<property>` elements:

Property	Mandatory	Description/Value
<code>ComponentType</code>	Yes	<code>com.sapportals.portal.layout</code>
<code>com.sap.portal.pcm.Title</code>	No	The display name of the layout
<code>com.sap.portal.pcm.Description</code>	No	A layout description
<code>com.sap.portal.reserved.layout.TemplateFile</code>	Yes	The name of the JSP page that defines the layout (relative to the <code>PORTAL-INF\jsp</code> directory)
<code>com.sap.portal.reserved.layout.Cont1</code> , <code>com.sap.portal.reserved.layout.Cont2</code> , and so forth	Yes	Each container defined in the JSP must have a property in the <code>portalapp.xml</code> . The names of these properties should end in <code>ContX</code> , where <code>X</code> is a sequence number starting at 1. The value is a name for the container.
<code>com.sap.portal.reserved.layout.TagLibLayout</code>	Yes	<code>/SERVICE/com.sap.portal.pagebuilder/taglib/layout.tld</code>

Modifying the Desktop and Navigation

<code>com.sap.portal.reserved.layout.TagLibHtmlb</code>	Yes	/SERVICE/com.sap.portal.htmlb/taglib/htmlb.tld Only required if using the default iView tray or adding HTMLB tags to the JSP page.
---	-----	---

For each container profile property described above, you can include the following meta-properties:

Property	Mandatory	Description
<code>plainDescription</code>	Yes	The display name of the container
<code>orientation</code>	Yes	Container orientation, which must be set to <code>vertical</code>
<code>designClass</code>	No	A CSS class for the HTML table created by the container (<code><table class="..."></code>)

3. Create a PAR file with your JSP pages and deployment descriptor. Place the JSP pages in the `PORTAL-INF/jsp` directory.
4. Deploy the PAR file.

3.7.2.1.1 Layout Tag Library

Purpose

A layout is defined in a JSP page, in which you specify where on the page to display the page's iViews. These locations are called containers.

Each of a page's iViews are assigned to one of the page's containers, and are displayed in that container during runtime. Within each container, each iView is displayed within a tray, or frame. You can use the portal's default tray or create a custom tray for all the iViews in a container.

The layout tag library helps you to create custom page layouts by providing tags that enable you to:

- Specify where on the page to place containers.
- Define a custom iView tray. For a custom tray, tags enable you to create links for invoking standard tray functions, such as refreshing the iView or displaying help for the iView.

For a sample layout JSP page, see [Sample Layout \[Page 33\]](#).



It may be easiest to take an existing layout and modify it. The default layouts are contained in the `com.sap.portal.layouts.default` application.

3.7.2.1.1.1 Types of Tags

This section describes the different groups of layout tags.

Basic Layout Tags

The following tags enable you to specify the location of the containers within the JSP page.

- [template \[Page 33\]](#): Specifies the start and end of the layout.
- [container \[Page 33\]](#): Creates a layout container whose iViews use the standard tray.
The tag is nested in the `template` tag.
- [containerWithTrayDesign \[Page 33\]](#): Creates a layout container whose iViews use the tray defined in the body of the tag. In the body of this tag, use the tags described in Custom Tray Tags below.
The tag is nested in the `template` tag.

Custom Tray Tags

All of the following tags, nested in the `containerWithTrayDesign` tag, are used to define a custom iView tray for the iViews displayed in the container.

- [iViewContent \[Page 33\]](#): Indicates where to display the iView's content within the tray.
- [iViewTitle \[Page 33\]](#): Displays the iView's title.
 - [IfiViewNameAvailable \[Page 33\]](#): Includes its body if the iView is set to display its title in its tray.
 - [IfNotiViewNameAvailable \[Page 33\]](#): Includes its body if the iView is set to not display its title in its tray.
- [iViewToggleOpen \[Page 33\]](#): Creates a link that shows the iView's contents.
- [iViewToggleClose \[Page 33\]](#): Creates a link that hides the iView's contents.
- [IfShowTray \[Page 33\]](#): Includes its body if the current iView is set to be displayed in a tray.
- [IfNotShowTray \[Page 33\]](#): Includes its body if the current iView is set to not be displayed in a tray.
- [iViewFamily \[Page 33\]](#): Displays the value of the iView's *Family* attribute, which indicates the type of content to be displayed in the iView.
- [iViewTrayColor \[Page 33\]](#): Displays the color value for the iView's *Family* attribute, which is generally used for the iView tray background color.

Tray Function Tags

The tag library includes a set of tags to create links for executing standard tray functions. For each type of link, there is one tag for creating the link anchor (for example, `iViewAbout`) and another for displaying the default link text translated for the current user (for example, `iViewAboutTitle`).

The following tags are available:

- **iViewAbout, iViewAboutTitle**: Creates a link that displays information about the iView.
- **iViewExpand, iViewExpandTitle**: Creates a link that opens the iView in a new window.
- **iViewHelp, iViewHelpTitle**: Creates a link that displays help for the iView.
- **iViewPersonalize, iViewPersonalizeTitle**: Creates a link that displays the personalize dialog for the iView.
- **iViewRefresh, iViewRefreshTitle**: Creates a link that refreshes the iView.

Modifying the Desktop and Navigation

- **IViewRemove, IViewRemoveTitle:** Creates a link that removes the iView from the current page.

For each type of link, there are also tags that enable you to check if the administrator has set the link to be displayed for the iView. There is one tag whose body is included if the link is set to be displayed, and another whose body is included if the link is set to not be displayed.

The following tags are available:

- **IfIViewAboutAvailable, IfIViewAboutNotAvailable**
- **IfIViewExpandAvailable, IfIViewExpandNotAvailable**
- **IfIViewHelpAvailable, IfIViewHelpNotAvailable**
- **IfIViewPersonalizeNotAvailable, IfIViewPersonalizeNotAvailable**
- **IfIViewRefreshAvailable, IfIViewRefreshNotAvailable**
- **IfIViewRemoveAvailable, IfIViewRemoveNotAvailable**

For more information on these tags, see [Tray Function Tags \[Page 33\]](#).

3.7.2.1.1.2 How to Use the Tag Library

This section describes the steps for creating from scratch a layout JSP page.



It may be easiest to take an existing layout and modify it. The default layouts are contained in the `com.sap.portal.layouts.default` application.

Procedure

1. Add a taglib directive for the layout tag library. JSP pages for the portal's default layouts use the tag name prefix `lyt`.

```
<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibLayout"
prefix="lyt" %>
```

If the JSP page includes HTMLB tags, add a reference to the HTMLB tag library.

2. Add a `template` tag, which indicates the start and end of the layout.

```
<lyt:template>

...

</lyt:template>
```

3. In the `template` tag, add HTML to divide the page into sections.
4. In the `layout` tag, add `container` or `containerWithTrayDesign` tags to indicate where you want iViews to be placed.

For example, the following uses HTMLB tags to divide the page into two equal columns and to add containers to each column.

```
<hbj:content id="myContext" >
  <hbj:page title="Portal Page">
    <hbj:gridLayout id="GridLayout1" width="100%"
cellSpacing="2">
```

Modifying the Desktop and Navigation

```

width="50%"      <hbj:gridLayoutCell rowIndex="1" columnIndex="1"
                  verticalAlignment="top">
                    <lyt:container id="column1" />
                  </hbj:gridLayoutCell>
width="50%"      <hbj:gridLayoutCell rowIndex="1" columnIndex="2"
                  verticalAlignment="top">
                    <lyt:container id="column2" />
                  </hbj:gridLayoutCell>
                </hbj:gridLayout>
            </hbj:page>
        </hbj:content>

```

The `container` tag creates a container that displays iViews with the default iView tray. The `containerWithTrayDesign` tag enables you to create a custom tray for the iViews within the container.

- For the `containerWithTrayDesign` tag, add HTML inside the tag's body to create a tray. Within the tray, indicate with the `iViewContent` tag where to place the iView's contents, and use other layout tags to create links to invoke tray functions.

For an example of a layout JSP page that creates a simple iView tray, see [Sample Layout \[Page 33\]](#).

3.7.2.1.1.3 Tag Reference

This section lists all the tags in the layout tag library.

3.7.2.1.1.3.1 container

Creates a container that displays the default HTMLB-based tray for all iViews in the container.

Attributes

Name	Mandatory	Description
id	Yes	The ID of the container. In the <code>portalapp.xml</code> for the PAR that includes the layout, this ID must be specified in a profile property called <code>com.sap.portal.reserved.layout.Cont<X></code> . For more information, see How to Create a Custom Layout [Page 33] .

Coordinating Tags

The tag must be nested in the following tag:

- [template \[Page 33\]](#)

Example

The following creates a layout with two columns of equal size. All iViews in the containers are displayed with the default iView tray.

Modifying the Desktop and Navigation

```

<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibHtmlb"
    prefix="hbj" %>
<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibLayout"
    prefix="lyt" %>

<lyt:template>
    <hbj:content id="myContext" >
        <hbj:page title="Portal Page">
            <hbj:gridLayout id="GridLayout1" width="100%"
cellSpacing="2">
                <hbj:gridLayoutCell rowIndex="1" columnIndex="1"
width="50%" verticalAlignment="top">
                    <lyt:container id="column1" />
                </hbj:gridLayoutCell>
                <hbj:gridLayoutCell rowIndex="1" columnIndex="2"
width="50%" verticalAlignment="top">
                    <lyt:container id="column2" />
                </hbj:gridLayoutCell>
            </hbj:gridLayout>
        </hbj:page>
    </hbj:content>
</lyt:template>

```

See Also

- [containerWithTrayDesign \[Page 33\]](#)

3.7.2.1.1.3.2 containerWithTrayDesign

Creates a container and enables you to create a custom tray for all iViews displayed in the container.



If you want the same tray for all iViews in all containers in the layout, place the code that defines the tray (that is, the body of the `containerWithTrayDesign` tag) in an include file, and then include the file in all your containers.

Attributes

Name	Mandatory	Description
id	Yes	The ID of the container. In the <code>portalapp.xml</code> for the PAR that includes the layout, this ID must be specified in a profile property called <code>com.sap.portal.reserved.layout.Cont<X></code> . For more information, see How to Create a Custom Layout [Page 33] .

Coordinating Tags

The tag must be nested in the following tag:

Modifying the Desktop and Navigation

- [template \[Page 33\]](#)

Example

The following creates a container with a custom tray that displays the title of the iView at the top, and then a set of links for invoking built-in tray functions. The iView's content is displayed after the links.

```
<lyt:containerWithTrayDesign id="column1">
  <TABLE border=1 cellspacing=10 cellpadding=10 style='WIDTH=100%;
  font-size:8.5pt;font-family:Tahoma;border-collapse:collapse;
  border:none;'>
    <TR>
      <TD>
        <b><lyt:IViewTitle/></b><br>
        <lyt:IViewExpand>Open in New
Window</lyt:IViewExpand>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewRefresh>Refresh</lyt:IViewRefresh>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewToggleOpen>Open</lyt:IViewToggleOpen>

        <lyt:IViewToggleClose>Close</lyt:IViewToggleClose>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewAbout>About</lyt:IViewAbout>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewHelp>Help</lyt:IViewHelp>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewRemove>Remove</lyt:IViewRemove>&nbsp;|&nbsp;&nbsp;&nbsp;
        <lyt:IViewPersonalize>Personalize</lyt:IViewPersonalize>
        <br>
        <lyt:IViewContent/>
      </TD>
    </TR>
  </TABLE>
</lyt:containerWithTrayDesign>
```

See Also

- [container \[Page 33\]](#)

3.7.2.1.1.3.3 IfIViewNameAvailable, IfNotIViewNameAvailable

Includes its body if the current iView's Show Object Name in Tray property is set to true. This property is set by a content administrator.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

See Also

- [IViewTitle \[Page 33\]](#)

Modifying the Desktop and Navigation

3.7.2.1.1.3.4 IfShowTray, IfNotShowTray

Includes its body if the current `iView`'s `Show Tray` property is set to `true`. This property is set by a content administrator.



If you use these tags, make sure to include both an `IfShowTray` and `IfNotShowTray` tag. In addition, make sure to include the `iViewContent` tag in both the `IfShowTray` and `IfNotShowTray` tag, or outside either tag; otherwise, the `iView`'s content may not be displayed in all cases.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

The following displays a custom `iView` tray if the current `iView` is set to be displayed in a tray.

Note that the `iViewContent` tag is outside both the `IfShowTray` and `IfNotShowTray` tags.

```
<lyt:containerWithTrayDesign id="column1">
  <TABLE border=1 cellspacing=10 cellpadding=10
style='WIDTH=100%;HEIGHT=100%;font-size:8.5pt;font-
family:Tahoma;border-collapse:collapse;border:none; '>
  <TR>
    <TD>
      <lyt:IfShowTray>
        <b><lyt:IViewTitle/></b><br>
        <lyt:IViewExpand>Open in New
Window</lyt:IViewExpand>
        &nbsp;|&nbsp;
        <lyt:IViewRefresh>Refresh</lyt:IViewRefresh>
        &nbsp;|&nbsp;
        <lyt:IViewToggleOpen>Open</lyt:IViewToggleOpen>

<lyt:IViewToggleClose>Close</lyt:IViewToggleClose>
        &nbsp;|&nbsp;

<lyt:IViewAbout>About</lyt:IViewAbout>&nbsp;|&nbsp;
        <lyt:IViewHelp>Help</lyt:IViewHelp>&nbsp;|&nbsp;

<lyt:IViewRemove>Remove</lyt:IViewRemove>&nbsp;|&nbsp;

<lyt:IViewPersonalize>Personalize</lyt:IViewPersonalize>
      </lyt:IfShowTray>

      <lyt:IfNotShowTray>

        ...

      </lyt:IfNotShowTray>

      <BR>
      <lyt:IViewContent/>
    </TD>
  </TR>
</TABLE>
</lyt:containerWithTrayDesign>
```

```

</TD>
</TR>
</TABLE>
</lyt:containerWithTrayDesign>

```

3.7.2.1.1.3.5 IViewContent

Indicates where to display the iView's content within a custom tray.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

The following creates a container with a custom tray that displays the title of the iView at the top, and then a set of links for invoking built-in tray functions. The iView's content is displayed after the links.

[illegible]

3.7.2.1.1.3.6 IViewFamily

Displays the value of the iView's *Family* attribute.

Modifying the Desktop and Navigation

The *Family* attribute is generally used to indicate the type of content contained in the iView, such as *Human Resources* or *Sales*, and to assign a color to the iView tray via the `iViewTrayColor` tag.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

```
<lyt:template>
  <lyt:containerWithTrayDesign id="column1">
    <div style="HeaderBox">
      <table><tr><td bgcolor=<lyt:iViewTrayColor/>>
        <lyt:iViewToggleOpen><lyt:iViewFamily/> -
        <lyt:iViewTitle/></lyt:iViewToggleOpen>
        <lyt:iViewToggleClose><lyt:iViewFamily/>
        </lyt:iViewToggleClose>&nbsp;&nbsp;&nbsp;<lyt:iViewExpand>
        </lyt:iViewExpand>
      </td></tr></table>
    </div>
    <lyt:iViewContent/>
  </lyt:containerWithTrayDesign>
</lyt:template>
```

See Also

- [iViewTrayColor \[Page 33\]](#)

3.7.2.1.1.3.7 iViewTitle

Displays the value of the `com.sap.portal.pcm.Title` property of the iView.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

The following creates a container with a custom tray that displays the title of the iView at the top, and then a set of links for invoking built-in tray functions.

```
<lyt:containerWithTrayDesign id="column1">
  <TABLE border=1 cellspacing=10 cellpadding=10 style='WIDTH=100%;
  font-size:8.5pt;font-family:Tahoma;border-collapse:collapse;
  border:none;mso-border-alt: solid navy .75pt;mso-yfti-
tbllook:480;
  mso-padding-alt:0in 5.4pt 0in 5.4pt;
  mso-border-insidev:.75pt solid navy'>
  <TR>
    <TD>
      <b><lyt:iViewTitle/></b><br>
```

Modifying the Desktop and Navigation

```

        <lyt:IViewExpand>Open in New
Window</lyt:IViewExpand>&nbsp;|&nbsp;&nbsp; 
        <lyt:IViewRefresh>Refresh</lyt:IViewRefresh>&nbsp;|&nbsp;&nbsp; 
        <lyt:IViewToggleOpen>Open</lyt:IViewToggleOpen>

<lyt:IViewToggleClose>Close</lyt:IViewToggleClose>&nbsp;|&nbsp;&nbsp; 
<lyt:IViewAbout>About</lyt:IViewAbout>&nbsp;|&nbsp;&nbsp; 
<lyt:IViewHelp>Help</lyt:IViewHelp>&nbsp;|&nbsp;&nbsp; 
<lyt:IViewRemove>Remove</lyt:IViewRemove>&nbsp;|&nbsp;&nbsp; 
<lyt:IViewPersonalize>Personalize</lyt:IViewPersonalize>
<br>
        <lyt:IViewContent/>
        </TD>
</TR>
</TABLE>
</lyt:containerWithTrayDesign>

```

See Also

- [IfViewNameAvailable, IfNotViewNameAvailable \[Page 33\]](#)

3.7.2.1.1.3.8 IViewToggleOpen, IViewToggleClose

Creates a link to hide or display the iView's contents.

The link for hiding the contents is only displayed if the contents are currently displayed. The link for displaying the contents is only displayed if the contents are currently hidden.

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

The following creates a container with a custom tray that displays the title of the iView at the top, and then a set of links for invoking built-in tray functions.

Note that only the Open or Close link will be created, and only one separator is written for both tags.

```

<lyt:containerWithTrayDesign id="column1">
  <TABLE border=1 cellspacing=10 cellpadding=10 style='WIDTH=100%;
font-size:8.5pt;font-family:Tahoma;border-collapse:collapse;
border:none;mso-border-alt: solid navy .75pt;mso-yfti-
tbllook:480;
mso-padding-alt:0in 5.4pt 0in 5.4pt;
mso-border-insidev:.75pt solid navy'>
  <TR>
    <TD>
      <b><lyt:IViewTitle/></b><br>
      <lyt:IViewExpand>Open in New
Window</lyt:IViewExpand>&nbsp;|&nbsp;&nbsp; 
      <lyt:IViewRefresh>Refresh</lyt:IViewRefresh>&nbsp;|&nbsp;&nbsp; 
      <lyt:IViewToggleOpen>Open</lyt:IViewToggleOpen>

```

Modifying the Desktop and Navigation

```

<lyt:IViewToggleClose>Close</lyt:IViewToggleClose>&nbsp;|&nbsp;&nbsp; 
    <lyt:IViewAbout>About</lyt:IViewAbout>&nbsp;|&nbsp;&nbsp; 
    <lyt:IViewHelp>Help</lyt:IViewHelp>&nbsp;|&nbsp;&nbsp; 
    <lyt:IViewRemove>Remove</lyt:IViewRemove>&nbsp;|&nbsp;&nbsp; 
    <lyt:IViewPersonalize>Personalize</lyt:IViewPersonalize>
    <br>
    <lyt:IViewContent/>
    </TD>
</TR>
</TABLE>
</lyt:containerWithTrayDesign>

```

3.7.2.1.1.3.9 IViewTrayColor

Displays the color value for the iView's *Family* attribute.

The color values are generally defined in the portal core iView on which all iViews are based (*Portal Content* → *Content Supplied by SAP* → *Core Objects* → *Core iView*, or by object ID *portal_content* → *com.sap.pct* → *default_objects* → *com.sap.portal.default_iView*).

This core iView defines the attribute *Family* (*com.sap.portal.iView.family*).

For the *Family* attribute, the following meta-attributes are defined:

- *validvalues*: A list of values for this attribute (*Group_0*, *Group_1*, and so forth)
- *validvalueTitle*<n>: The display title for each valid value. One *validvalueTitle* meta-attribute is defined for each valid value in the *validvalues* meta-attribute (*validvalueTitle0*, *validvalueTitle1*, and so forth)
- *Color*<n>: The color for each valid value. One *Color* meta-attribute is defined for each valid value in the *validvalues* meta-attribute (*Color0*, *Color1*, and so forth)
- To set the display name of each iView family name, set the *validvalueTitle* meta-attribute for each family name.
- To set the color for each iView family name, set the *Color* meta-attribute for each family name.

Setting Color for iView Family

- For example, to create a group for human resources iViews, and to display these iViews with a red iView tray, set the following meta-attributes for the *Family* attribute of the core iView:
 - *validvalues*: Designate one of the valid values, such as *Group_4*, for human resources iViews.
By default, values *Group_0* to *Group_9* are defined. If you need, add additional values, such as *Group_10*, *Group_11* and so forth.
 - *validvalueTitle4*: Set to **Human Resources**.
 - *Color4*: Set to **#FF0000**.
- Use the PCD Inspector tool to set the values. For more information on the PCD Inspector, see [PCD Inspector \[External\]](#).

Modifying the Desktop and Navigation

Coordinating Tags

The tag must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

```
<lyt:template>
  <lyt:containerWithTrayDesign id="column1">
    <div style="HeaderBox">
      <table><tr><td bgcolor="<lyt:IViewTrayColor/>">
        <lyt:IViewToggleOpen><lyt:IViewFamily/> -
        <lyt:IViewTitle/></lyt:IViewToggleOpen>
        <lyt:IViewToggleClose><lyt:IViewFamily/>
        </lyt:IViewToggleClose>&nbsp;&nbsp;&nbsp;<lyt:IViewExpand>
        </lyt:IViewExpand>
      </td></tr></table>
    </div>
    <lyt:IViewContent/>
  </lyt:containerWithTrayDesign>
</lyt:template>
```

See Also

- [IViewFamily \[Page 33\]](#)

3.7.2.1.1.3.10 template

Indicates the start and end of the layout.

Variables

- `epPageVariables`: Enables you to get information about the current client, and whether the layout is being used in design time or runtime.

The variable has the following two methods:

- `epPageVariables.getRunMode()`: Returns either `RunMode.RUN_TIME` or `RunMode.DESIGN_TIME`
- `epPageVariables.getUserAgent()`: Returns an object of type `IUserAgent`, which enables you to find out about the client, such as the browser type and version.

For more information, see [User Agent Service \[Page 33\]](#).

The scope of `epPageVariables` is from the beginning of the `template` tag to the end of the JSP page.

Example

The following is a layout with one container that executes unspecified code if the layout is displayed in runtime and other code if the client is Internet Explorer.

Modifying the Desktop and Navigation

```

<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibHtmlb"
prefix="hbj" %>
<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibLayout"
prefix="lyt" %>

<%@ page import="com.sapportals.portal.useragent.IUserAgent" %>
<%@ page import="com.sapportals.portal.useragent.IUserAgentConstants"
%>
<%@ page import="com.sapportals.portal.pb.layout.taglib.variabledef
.RunMode" %>

<%
RunMode pageRunMode;
IUserAgent pageUserAgent;
String pageUserAgentType;
%>
<lyt:template>
<%
    //Resolve page RunMode
    pageRunMode = epPageVariables.getRunMode();

    //Resolve page UserAgent
    pageUserAgent = epPageVariables.getUserAgent();
    pageUserAgentType = pageUserAgent.getType();
%>
    <hbj:content id= "myContext">
        <hbj:page title="Portal Page">
<%
            if(pageRunMode == RunMode.RUN_TIME) {

                ...

            }

if(pageUserAgentType.equals(IUserAgentConstants.UA_TYPE_MSIE)) {

                ...

            }
%>
            <lyt:container id="the_container_id" />
        </hbj:page>
    </hbj:content>
</lyt:template>< /p>

```

3.7.2.1.1.3.11 Tray Function Tags

The layout tag library provides tags for creating links for invoking tray functions, such as refreshing or personalizing the iView.

For each type of link, the following types of tags are available (an example is shown in parentheses):

- **Anchor Tag** (*IViewAbout*): Creates a link to invoke a tray function. The tag's body is the text or image for the link.

Modifying the Desktop and Navigation

- **Anchor Text Tag** (`IViewAboutTitle`): Displays default and localized text for the link to invoke a tray function. This tag is generally included in the body of the anchor tag.
- **Conditional, Administrator Setting is On** (`IfIViewAboutAvailable`): The body of the tag is included if the iView's properties are set to display the specific link. For each iView, the administrator can choose whether to display links to specific tray functions.
- **Conditional, Administrator Setting is Off** (`IfIViewAboutNotAvailable`): The body of the tag is included if the iView's properties are set not to display the specific link. For each iView, the administrator can choose whether to display links to specific tray functions.

Tray Functions

Tags are available for the following tray functions (available tags are listed after each function):

- **About:** Displays information about the iView.
 - `IViewAbout`
 - `IViewAboutTitle`
 - `IfIViewAboutAvailable`
 - `IfIViewAboutNotAvailable`
- **Expand:** Opens the iView in a new window.
 - `IViewExpand`
 - `IViewExpandTitle`
 - `IfIViewExpandAvailable`
 - `IfIViewExpandNotAvailable`
- **Help:** Displays help for the iView.
 - `IViewHelp`
 - `IViewHelpTitle`
 - `IfIViewHelpAvailable`
 - `IfIViewHelpNotAvailable`
- **Personalize:** Displays the personalize dialog for the iView.
 - `IViewPersonalize`
 - `IViewPersonalizeTitle`
 - `IfIViewPersonalizeNotAvailable`
 - `IfIViewPersonalizeNotAvailable`
- **Refresh:** Refreshes the iView.
 - `IViewRefresh`
 - `IViewRefreshTitle`
 - `IfIViewRefreshAvailable`
 - `IfIViewRefreshNotAvailable`

Modifying the Desktop and Navigation

- **Remove:** Removes the iView from the current page.
 - IViewRemove
 - IViewRemoveTitle
 - IfIViewRemoveAvailable
 - IfIViewRemoveAvailable

Coordinating Tags

These tags must be nested in the following tag:

- [containerWithTrayDesign \[Page 33\]](#)

Example

The following is a section of a layout JSP page that displays a link for each tray function if the administrator has set that link to be displayed for the current iView. The code must be nested in a `containerWithTrayDesign` tag.

```
<lyt:IfIViewExpandAvailable>
  <lyt:IViewExpand><lyt:IViewExpandTitle/>
</lyt:IViewExpand>&nbsp;|&nbsp;
</lyt:IfIViewExpandAvailable>

<lyt:IfIViewRefreshAvailable>
  <lyt:IViewRefresh><lyt:IViewRefreshTitle/>
</lyt:IViewRefresh>&nbsp;|&nbsp;
</lyt:IfIViewRefreshAvailable>

<lyt:IfIViewAboutAvailable>
  <lyt:IViewAbout><lyt:IViewAboutTitle/>
</lyt:IViewAbout>&nbsp;|&nbsp;
</lyt:IfIViewAboutAvailable>

<lyt:IfIViewHelpAvailable>
  <lyt:IViewHelp><lyt:IViewHelpTitle/>
</lyt:IViewHelp>&nbsp;|&nbsp;
</lyt:IfIViewHelpAvailable>

<lyt:IfIViewRemoveAvailable>
  <lyt:IViewRemove><lyt:IViewRemoveTitle/>
</lyt:IViewRemove>&nbsp;|&nbsp;
</lyt:IfIViewRemoveAvailable>

<lyt:IfIViewPersonalizeAvailable>
  <lyt:IViewPersonalize><lyt:IViewPersonalizeTitle/>
</lyt:IViewPersonalize>
</lyt:IfIViewPersonalizeAvailable>
```

Modifying the Desktop and Navigation

3.7.2.2 Sample Layout

The following is a sample JSP page and `portalapp.xml` file for a PAR file that defines a custom layout.

The layout includes one container whose iViews are displayed with a custom tray. The tray shows the title of the iView at the top, and then a link to either hide or display the contents.

A set of links is displayed just below the title for invoking standard tray functions. Each button is only displayed if the administrator has set the link to be displayed in the iView's properties. The iView's content is displayed just below.

JSP Page

```
<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibHtmlb"
prefix="hbj" %>
<%@ taglib
uri="prt:taglib:com.sap.portal.reserved.layout.TagLibLayout"
prefix="lyt" %>
<lyt:template>
  <hbj:content id="myContext" >
    <hbj:page title="Portal Page">
      <lyt:containerWithTrayDesign id="the_container_id">

        <TABLE border=1 cellspacing=10 cellpadding=10
style='WIDTH=100%;font-size:8.5pt;font-family:Tahoma;border-
collapse:collapse;border:none;'>
          <TR>
            <TD>
              <b><lyt:IViewTitle/></b>&nbsp;|

              <lyt:IViewToggleOpen>&gt;&gt;&gt;</lyt:IViewToggleOpen>

              <lyt:IViewToggleClose>&lt;&lt;&lt;&lt;</lyt:IViewToggleClose>
                <BR><BR>
                <lyt:IfIViewExpandAvailable>
                  <lyt:IViewExpand><lyt:IViewExpandTitle/>
                </lyt:IViewExpand>
                &nbsp;|&nbsp;
                </lyt:IfIViewExpandAvailable>
                <lyt:IfIViewRefreshAvailable>

              <lyt:IViewRefresh><lyt:IViewRefreshTitle/>
              </lyt:IViewRefresh>
              &nbsp;|&nbsp;
              </lyt:IfIViewRefreshAvailable>
              <lyt:IfIViewAboutAvailable>
                <lyt:IViewAbout><lyt:IViewAboutTitle/>
              </lyt:IViewAbout>
              &nbsp;|&nbsp;
              </lyt:IfIViewAboutAvailable>
              <lyt:IfIViewHelpAvailable>
                <lyt:IViewHelp><lyt:IViewHelpTitle/>
              </lyt:IViewHelp>
              &nbsp;|&nbsp;
              </lyt:IfIViewHelpAvailable>
              <lyt:IfIViewRemoveAvailable>
```


Modifying the Desktop and Navigation

```
<lyt:IViewRemove><lyt:IViewRemoveTitle/>  
    </lyt:IViewRemove>  
    &nbsp;|&nbsp;  
    </lyt:IfIViewRemoveAvailable>  
    <lyt:IfIViewPersonalizeAvailable>  
  
    <lyt:IViewPersonalize><lyt:IViewPersonalizeTitle/>  
  
    </lyt:IViewPersonalize>  
  
    </lyt:IfIViewPersonalizeAvailable>  
        <br>  
        <lyt:IViewContent/>  
    </TD>  
    </TR>  
    </TABLE>  
  
    </lyt:containerWithTrayDesign>  
    </hbj:page>  
    </hbj:content>  
</lyt:template>
```

portalapp.xml

The text in **bold** is customizable. You can also add references to other portal services, if necessary.

The value for the property called `com.sap.portal.reserved.layout.Cont1` is the same as the value for the `id` attribute for the `container` or `containerWithTrayDesign` tag for that column in the JSP page.

```
<application>
  <application-config>
    <property name="SharingReference" value="com.sap.portal.htmlb,
      com.sap.portal.pagebuilder"/>
  </application-config>
  <components>
    <component name="myFullWidthLayout">
      <component-config>
        <property name="ClassName"
value="com.sapportals.portal.pb.layout.PageLayout"/>
        <property name="ResourceBundleName"
          value="pagebuilder_nls"/>
      </component-config>
      <component-profile>
        <property name="ComponentType"
          value="com.sapportals.portal.layout" />
        <property name="com.sap.portal.pcm.Title"
          value="My 1 Column Layout (Full Width)"/>
        <property name="com.sap.portal.pcm.Description"
          value="Layout displaying one full-width column"/>
      </property
name="com.sap.portal.reserved.layout.TagLibLayout"
value="/SERVICE/com.sap.portal.pagebuilder/taglib/layout.tld"/>
```

Modifying the Desktop and Navigation

```

        <property
name="com.sap.portal.reserved.layout.TagLibHtmlb"
        value="/SERVICE/com.sap.portal.htmlb/taglib/htmlb.tld "/>
        </property>

name="com.sap.portal.reserved.layout.TemplateFile"
        value="myFullWidth.jsp"/>
        <property name="com.sap.portal.reserved.layout.Cont1"
        value="the_container_id">
            <property name="title" value="my container"/>
            <property name="orientation" value="vertical"/>
        </property>
    </component-profile>
</component>
</components>
</services>
</application>

```

The HTMLB sharing reference is only needed if you use the default iView tray or you add HTMLB code to the JSP page.

3.7.3 Object-Based Navigation

Purpose

To provide the knowledge necessary for portal content developers to create iViews using the capabilities of object-based navigation.

Object-based navigation (OBN) offers portal users an additional method of navigation, which is role-dependent and based on business objects from productive back-end systems.

OBN is based on a structure of business objects having one or more operations attached to them, where each operation can have one or more iViews to implement it. Each operation has a priority, and selecting an OBN link in an iView does one of the following:

- executes the default operation, the one with the highest priority (if the operation has an implementing iView for the role of the user)
- presents a context menu displaying all of the operations attached to the main business object of the source iView (where those operations have implementing iViews for the role of the user)

The primary capability offered by OBN is that the iView returning data to the user, during navigation, is accessed dynamically during runtime, based on the user's role. In other words, two different users may perform the same navigation operation, clicking the same link from the same iView, and the data returned to each of them will be different because of role dependency.

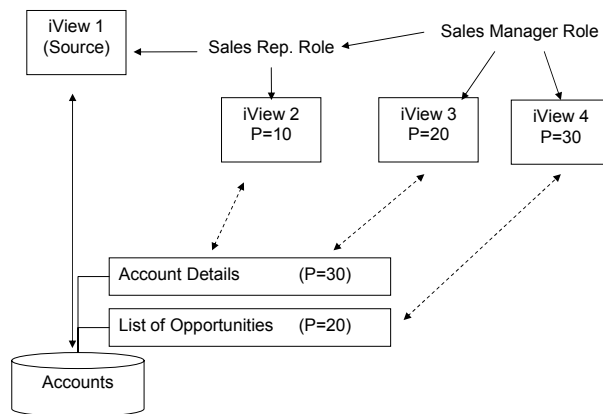


Familiarity with the standard documentation of object-based navigation is recommended. See help.sap.com/nw04 → *<language>* → SAP Library → SAP NetWeaver → People Integration → Portal → Administration Guide → Content Administration → Navigation → Object-Based Navigation.

Use Case

This use case deals with the implementation of the default operation.

Modifying the Desktop and Navigation



Two users in a company with separate roles, a sales representative and a sales manager, each choose a link in an iView representing a specific account, based on the same business object. This business object has several operations attached to it, one of which is *Account Details*, another being *List of Opportunities*. The sales representative only has permission to see an iView that implements *Account Details*. The sales manager is authorized to see the iViews implementing both operations. However, for the Sales Manager role, the operation *List of Opportunities* has a higher priority.

3.7.3.1 Glossary

Glossary of terms related to object-based navigation

Business Object	A Portal Platform entity representing a business objects of a back end system
Source iView	The iView from which an OBN operation is activated
Implementing iView / Page	Any iView or page attached to a business object operation
Priority	A value specifying the default operation to be implemented for a specific business object (The higher value has the higher priority.)
Business Object Operation (operation)	A portal entity used by an administrator to define a connection between a business object and implementing iViews
Role	The largest semantic unit within the portal content objects, a role is a folder hierarchy comprising other content objects (worksets, pages, iViews). Roles are assigned to users, meaning that users can only access the content of their specific role or roles
Relation	A defined logical link between two business objects, usually manifested by identical properties
HRNP Link	A UI reference that, when activated (for example, by a mouse click), implements the Hyperrelational Navigation Protocol (HRNP), which enables Drag&Relate navigation

3.7.3.2 Defining the End-User Experience

There are two ways for the end user to activate OBN during runtime, an iView link and a context menu.

iView Link

A link, representing an OBN call, is developed on top of a business object operation and appears in the source iView. Clicking the link opens the default implementing iView, in other words, the iView implementing the business object operation with the highest priority in the user role.

To display the default implementing iView, the object-based navigation service searches as follows:

- First for the operation with the highest priority, if more than one operation is associated with the main object of the source iView.
- If the high-priority operation has associated with it more than one implementing iView, the OBN service selects the iView for which the current operation has the highest priority.

Information for the Developer

Create an OBN call on the client side using the API `doObjBasedNavigate`, which is available from Enterprise Portal Client Manager (EPCM), or encapsulate this call in Java in the iView code. For more information, see [The OBN Call \[Page 33\]](#).

Context Menu

It is also possible to access a context menu on top of a business object operation.

In this case, there are several operations associated with the main object of the source iView and choosing the OBN arrow icon in the iView displays a subset of the operations. Upon choosing the desired operation, the relevant implementing iView is displayed.

Logic behind the Context Menu

- An operation can appear in a context menu only once; therefore, in the event that several iViews in the user roles implement the same operation, the operation instance appearing in the context menu is the one whose iView-based priority is the highest (as assigned in the OBN Editor).

Only operations whose implementing iViews are contained in one of the user roles are displayed in the context menu.

- Using the Object-Based Navigation Editor, an administrator may customize the name of a specific operation. In such a case, the customization provided by the implementing iView appears in the context menu, overwriting the original name of the operation.

Although the content developer may use the supplied APIs to implement the context menu as desired, the logic described here cannot be changed.

Information for the Developer

Two levels of APIs are available for the content developer to create context menus for object-based navigation iViews.

- A high-level API is available for automatically creating context menus with the SAP standard UI look and feel.

Modifying the Desktop and Navigation

- Lower level APIs may be used as well to create context menus for which the UI is not supplied. Essentially, the content developer can create a context menu for any UI element and associate an OBN call with each operation to be include in the context menu list. The UI can be created as required by specific organizational standards.

An example of the use of the lower level APIs would be in BW reports, where the implementation of the context menu suits the appearance of the native interface.

3.7.3.3 Business Object Operation

Purpose

To explain the process of creating and configuring the business object operation.


The core element of OBN is the business object operation. The operation defines the connection between the business object and the iView that implements it. Multiple operations may be defined for the source business object and multiple implementing iViews may be attached to each operation. The operation may also be customized for any specific iView.

Prerequisites

- At least one portal system has been defined to represent back-end systems.
- Business objects have been imported to the Portal Content Directory (PCD). For detailed information, see the documentation for object-based navigation at help.sap.com → <language> → *SAP Library* → *SAP NetWeaver* → *People Integration* → *Portal* → *Administration Guide* → *Content Administration* → *Navigation* → *Object-Based Navigation*.

Workflow for Defining an Operation

Operation Definition for the Content Administrator

Step 1 Create operation	<p>OBN starts with the creation of an operation for a business object using the Business Object Editor, available from the context menu of any business object in the Content Catalog tree.</p>  <ul style="list-style-type: none"> • Once defined, the operation ID cannot be changed. • The priority value must be a non-negative integer. <p>Once the operation has been created and the business object saved, the business object can be considered as an OBN source object. iViews displaying data of such business objects may contain OBN calls.</p>
-----------------------------------	---

Modifying the Desktop and Navigation

Step 2 Implement operation	<p>In order to complete the OBN definition we must implement the operation by providing an iView to be launched when an OBN call is activated. Operations and iView are connected to each in two ways:</p> <ul style="list-style-type: none"> • When the Business Object Editor is open, attach iViews to operations, using the context menu of any iView in the Portal Content Catalog. • Or, for the modification of operations for specific iViews: when the Object-Based Navigation Editor for the iView is open, attach operations to the iView, using the context menu of any operation available in the Business Object folder. <p>The administrator is responsible for creating operations and attaching implementing iViews.</p>
Step 3 Modify an operation from an iView (optional)	<p>If it is necessary to override the existing priority or change the display name of an operation for a specific role, these modifications may be made in the Object-Based Navigation Editor for the specific iView.</p>
Step 4 Add OBN calls to OBN source iViews	<p>The next step in the OBN definition is the embedding of OBN calls into iViews that are intended to activate operations. These calls activate object-based navigation at runtime.</p> <p>iView developers are responsible for creating these calls in their iViews. For more information, see The OBN Call [Page 33].</p>

3.7.3.3.1 Implementing iViews

Implementation of operations is done by the attachment of iViews to operations. Each iView attached to an operation is the implementing iView of the operation.

Role of implementing iViews in OBN

The ability to have specific navigation customizations for different users according to their roles is a key feature of OBN. All portal users are assigned their roles according to their business interests and permissions. Since permissions to iViews are granted only those contained within the role of the user, OBN-specific customizations are achieved using role maintenance for the implementing iViews, and modifying the priorities of the business object operations to which they are attached.

Priorities of implementing iViews in OBN

Access to portal content is role-based, and a user may have more than one role. Additionally, an iView may have multiple operations associated with it, and an operation may be implemented by multiple iViews. In this situation, a user launching an iView may have quite a large pool of implementing iViews, presenting the portal runtime with conflicts as to which iView to implement.

The ability to edit the priority of these operations is provided within the context of implementing iViews. The operation priority defined within from implementing iView takes precedence over the priority assigned to it from the Business Object Editor.

Example

The following example is valid for the iView link scenario in which the default operation is activated.

Modifying the Desktop and Navigation

In the illustration, there are two users with two corresponding roles: Sales Representative and Sales Manager. The Sales Representative role has permissions for iViews 1 and 2, but not to 3 or 4. The Sales Manager Role has permissions for all iViews.

Note the following:

- iView 1 is the source iView containing the OBN call.
- iView 1 is based on the business object Accounts.
- The operations attached to the business object Accounts are Account Details and List of Opportunities; each operation displays the priority it was defined for it in the Business Object Editor.
- The P value represents the operation priority.
- iViews 2, 3, and 4 are implementing iViews, each displaying the modified priority value for the associated operations, in accordance with the role of the user.

Error! Objects cannot be created from editing field codes.

When the Sales Representative role chooses the OBN link in iView 1, only iView 2, implementing Account Details, is displayed, even though the priority of the Account Details operation was modified within the context of the iView, to be lower than the original priority assigned to the operation. This is the only available iView to this role for this operation.

When the Sales Manager Role chooses the OBN link in iView 1, iView 4, implementing List of Opportunities, is displayed. Even though this operation has a lower priority assigned to it originally, the priorities of the operations have been modified within the context of the iViews, so that for this role iView 4 takes precedence over iView 3.



It needs to be kept in mind that the search pattern of the object-based navigation service first to look for the operation with the highest priority, and then to look for the implementing iView (if the high-priority operation has associated with it more than one), for which the current operation has the highest priority.

3.7.3.3.2 The OBN Call

OBN calls retrieve only the implementing iView of the default operation; they do not implement the context menu UI. OBN calls may be embedded in the source code of the iViews in order to activate object-based navigation functionality. These calls can either be written to the client in JavaScript or wrapped within the code of the iView.

JavaScript OBN Call

The Enterprise Portal Client Manager (EPCM) offers an API named `doObjBasedNavigate` that serves as an OBN call and is embedded in iViews from which OBN to be activated.

To activate the API from your JavaScript write the following:

```
EPCM.DoObjBasedNavigate (systemAlias, businessObjName, objValue,
Operation)
```

Parameters of `doObjBasedNavigate`

Parameter	Optional	Description
systemAlias	False	System alias of business object

Modifying the Desktop and Navigation

<code>businessObjName</code>	False	Name of the business object for which the operation has been defined. Note that <code>businessObjName</code> and <code>systemAlias</code> together define the Distributed Query Engine (DQE) name of the business object.
<code>objValue</code>	True	Data to pass into target iView if the iView resulting from navigation should represent relative data. The <code>objValue</code> parameter can be any string that is added to the target iView URL after "?". Thus the writer of the target iView can access the <code>objValue</code> from the iView request.
<code>Operation</code>	True	Specifies the operation to produce in case several operations are available for the business object.

Java OBN Call

Java content developers may use `createObjectNavigateClientCall` API exposed by `com.sapportals.portal.navigation.NavigationService`. The purpose of the `createObjectNavigateClientCall` API is to encapsulate the `EPCM.DoObjBasedNavigate(...)` call for Java developers so that they need not be familiar with details of its syntax (which may also be subject to change). The parameters list for `createObjectNavigateClientCall` is identical to that of `DoObjBasedNavigate` of EPCM.

Script for Modification of objValue Parameter

The portal content administrator can modify the string value passed to the `objValue` parameter. This is done using the `OBNObjValueManipulation(objValue)` JavaScript function, which is executed before the implementing iView is called. The function body is exposed in the Object-Based Navigation Editor of the implementing iView.

Adapting Externally Developed Content

External components, which include OBN calls, were developed in a different portal against system aliases different from those used by the destination portal. These may be integrated by defining for the relevant systems additional aliases identical to those of the imported content.

Note that system alias is used to identify the system from which the business objects come.

3.7.3.4 Getting Started: OBN Examples

Use

This section describes the steps required to run the object-based navigation code samples provided with the PDK.

There is the `com.sap.portal.unification.OBNExamples` PAR file available in the Portal Platform. This PAR file contains examples that illustrate different uses of OBN and the steps for implementing them.

Modifying the Desktop and Navigation

Prerequisites

A running portal.

Procedure

In order to make the examples work, perform the following steps that follow.

1. Deploy the `com.sap.portal.unification.OBNEexamples` PAR file. The PAR file comes as a part of OBN PDK.
2. Create a system and its alias.
 - a. Create a JDBC Connector System on top of an SQL Server pubs database.
 - b. Define an alias for the system.
For our purposes, the alias name must be **OBNTTestSysAlias**.
 - c. Make sure there is user mapping for this alias.

3. In the portal, open the Business Object Importer and import the authors, titles, and titleauthor business objects from the OBNTTestSysAlias.

4. Create an OBNEexamples folder in the portal content tree.

You can use any name for the folder instead of OBNEexamples mentioned above.

This folder will hold iViews imported from the `com.sap.portal.unification.OBNEexamples` PAR file.

5. Import all the iViews from the PAR file into the folder.

The iViews are:

- WelcomeOp_iView
 - ParamPassOp_iView
 - ParamScriptOp_iView
 - DfltOpAuthors_iView
 - RelationOpAuthors_iView
 - ParamDisplay_iView
 - Titles_iView
6. Assign your user to **eu_role** (relevant for Sample 1 only).
 7. Create the roles **OBNEexamSrcRole** and **OBNEexamPIRole**, and assign the roles to your user.
 8. Create an **OBNSrcIViews** workset for OBNEexamSrcRole.
This workset will contain iViews from which to activate OBN for all the examples available in the `com.sap.portal.unification.OBNEexamples` PAR file.
 9. Add all the iViews listed below from the OBNEexamIViews folder to the workset. Add them as Delta Links:
 - WelcomeOp_iView
 - ParamPassOp_iView
 - ParamScriptOp_iView
 - DfltOpAuthors_iView
 - RelationOpAuthors_iView

Modifying the Desktop and Navigation

10. Create the workset **OBNImpIViews** workset for OBNEampIRole.

Place all operation-implementing iViews supplied by the `com.sap.portal.unification.OBNEamples` PAR file in this workset.

- ParamDisplay_iView
- Titles_iView



Set the Entry Point property of both the worksets to 'Yes' in order to permit seeing it at runtime.

11. Create operations and attach the implementing iViews to them as determined by the examples included in the PAR.

Create an operation under the authors business object of OBNTesSysAlias for each example available in the OBNEamples PAR file.

It is important to place the operation in the right place in the portal Content Catalog, and to keep their names as they appear the examples included in the PAR.

3.7.3.4.1 Example 1: Basic OBN

Use

Show object-based navigation based on the `com.sap.portal.unification.OBNEamples` PAR.

Sample Details:

Description	Activates the Welcome KM iView of the portal upon button-click
Operation name	WelcomeOp - defined for authors business object
Source iView	WelcomeOpIView
Target iView	<code>com.sap.portal.welcome_iView</code>

Procedure

1. Using the Business Object Editor, create an operation with the name *WelcomeOp* for the authors business object.
2. Attach the created operation to `com.sap.portal.km_welcome_iView`.

Operations and iViews may be attached to one another from either the Business Object Editor or the Object-Based Navigation Editor of a specific iView. In this case, however, since this iView does not appear in the Content Catalog, but is only accessible from the role of the user, the procedure is performed as follows, using the Object-Based Navigation Editor of the iView:

- a. In the Content Catalog tree, go to *Portal Content* → *Portal Users* → *Standard Portal Users* → *Standard Users Role*.
- b. From the context menu, choose *Edit Object*.
- c. In the editor, expand the *Welcome* workset.
- d. Select the *Welcome* page and choose the *Edit* button at the bottom of the editor.

Modifying the Desktop and Navigation

- e. Select the Knowledge Management & Collaboration iView and click the *Edit* button for it.
- f. In the editor drop-down list choose Object-Based Navigation to open the OBN Editor for this iView.
- g. In the Content Catalog, navigate to the desired operation as follows: *Business Objects* → *OBNTTestSysAlias* → *dbo* → *authors* → *WelcomeOp* operation.
- h. From the context menu of the operation, choose *Add Operation to iView*.
- i. Save the iView changes.

To activate these steps at runtime:

1. Go to the OBNSrcIViews workset and choose the WelcomeOpIView.
2. Choose the *Activate OBN* button.

The Knowledge Management & Collaboration iView is displayed as a result of the WelcomeOp operation.

3.7.3.4.2 Example 2: OBN with Parameter**Use**

Show object-based navigation, including passing a parameter value to the implementing iView, based on the `com.sap.portal.unification.OBNExamples` PAR.

Sample Details:

Description	Opens implementing iView upon button-click in source iView OBN call of passes value entered by user to the implementing iView. The implementing iView displays the parameter.
Operation name	ParamPassOp
Source iView	ParamPassOp_iView
Target iView	ParamDisplay_iView

Procedure

1. Create an operation with the name *ParamPassOp* for the authors business object.
2. Attach the created operation to *ParamDisplay_iView*.

Operations and iViews may be attached to one another from either the Business Object Editor or the Object-Based Navigation Editor of a specific iView. In this case, however, since this iView does not appear in the Content Catalog, but is only accessible from the role of the user, the procedure is performed as follows, using the Object-Based Navigation Editor of the iView:

- a. In the Content Catalog tree, go to *Portal Content* → *OBNExamIViews* → *OBNExamRoles* → *OBNExamImplRole*.
- b. From the context menu, choose *Edit Object*.
- c. In the editor, expand the *OBNImplIViews* workset.
- d. Select the *ParamDisplay_iView* and choose the *Edit* button at the bottom of the editor.

Modifying the Desktop and Navigation

- e. In the editor drop-down list choose Object-Based Navigation to open the OBN Editor for this iView.
- f. In the Content Catalog, navigate to the desired operation as follows: *Business Objects* → *OBNTTestSysAlias* → *dbo* → *authors* → *ParamPassOp* operation.
- g. From the context menu of the operation, choose *Add Operation to iView*.
- h. Save the iView changes.

To activate these steps at runtime:

1. Go to the *OBNSrc/Views* workset.
2. Choose *ParamPassOp_iView*.
3. Enter some string in the *Parameter to pass* item.
4. Click the *Activate OBN button*.

ParamDisplay_iView represents the typed-in "ParamPassOp_iView" string.

3.7.3.4.3 Example 3: OBN with Parameter Modification

Use

Show object-based navigation, including passing a parameter value to the implementing iView, based on the `com.sap.portal.unification.OBNEexamples` PAR.

Sample Details:

Description	<p>Opens implementing iView on button-click in source iView.</p> <p>OBN call passes value entered by user to the implementing iView and the parameter is changed in <code>OBNObjValueManipulation()</code> script method.</p> <p>The implementing iView displays the updated parameter.</p>
Operation name	<i>ParamScriptOp</i>
Source iView	<i>ParamScriptOp_iView</i>
Target iView	<i>ParamDisplay_iView</i>

Procedure

1. Create an operation with the name *ParamScriptOp* for the authors business object.
2. Attach the created operation to *ParamDisplay_iView*.

Operations and iViews may be attached to one another from either the Business Object Editor or the Object-Based Navigation Editor of a specific iView. In this case, however, since this iView does not appear in the Content Catalog, but is only accessible from the role of the user, the procedure is performed as follows, using the Object-Based Navigation Editor of the iView:

- a. In the Content Catalog tree, go to *Portal Content* → *OBNEexamIViews* → *OBNEexamRoles* → *OBNEexamImplRole*.
- b. From the context menu, choose *Edit Object*.
- c. In the editor, expand the *OBNImplIViews* workset.

Modifying the Desktop and Navigation

- d. Select the *ParamDisplay_iView* and choose the *Edit* button at the bottom of the editor.
- e. In the editor drop-down list choose Object-Based Navigation to open the OBN Editor for this iView.
- f. In the Content Catalog, navigate to the desired operation as follows: *Business Objects* → *OBNTTestSysAlias* → *dbo* → *authors* → *ParamPassOp* operation.
- g. From the context menu of the operation, choose *Add Operation to iView*.
- h. Save the iView changes.
- i. Update the `OBNObjValueManipulation()` method of the *ParamScriptOp*.
- j. Select *ParamScriptOp* in the Operations table of the iView.
- k. Change the existing script to the following JavaScript, which adds asterisks to the parameter passed into the function.

```
var paramStr;
var array = objValue.split("=");
if (array.length != 2)
{
    //Error: Unknown parameter format => leave the
    parameter without any change
    paramStr=objValue;
}
else
{
    var paramName = array[0];

    var paramVal = "***_";
    paramVal += array[1];
    paramVal += "_***";
    paramStr = paramName + "=" + paramVal;
}
return paramStr;
```

- l. Choose *Set*.
 - m. Save the iView changes.
- The JavaScript is set to the `OBNObjValueManipulation()` function of the *ParamScriptOp* operation.

To activate these steps at runtime:

1. Go to the *OBNSrc/Views* workset.
2. Choose *ParamScriptOp_iView*.
3. Enter some string in the *Parameter to pass* item.
4. Click the *Activate OBN* button.

Connecting to Backend Systems

5. The *ParamDisplay_iView* represents the *ParamPassOp_iView* string, enclosed with asterisks.

3.8 Connecting to Backend Systems

This section describes how to connect to back-end systems and applications, and includes the following:

- [Application Integrator \[Page 33\]](#)
- [Connector Framework \[Page 33\]](#)
- [Dynamic System Resolution \[Page 33\]](#)

3.8.1 Application Integrator

Purpose

With the application integrator service you can integrate remote Web applications into the portal. The service creates portal components that serve as template for iViews which represent the remote application.

Features

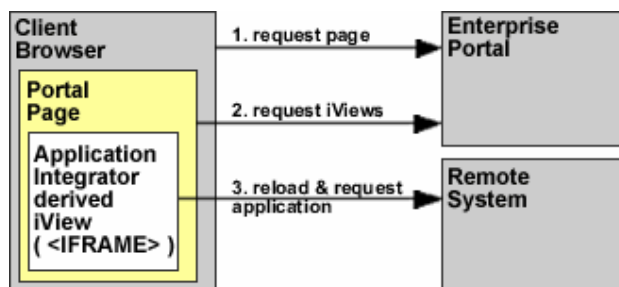
The application integrator service provides portal components, focused on SAP applications:

- SAP Transactions (SAP backend transactions)
- Internet Application Components (IAC)
- Business Server Pages (BSP)
- BEx Web Applications (former BW Report)
- Crystal Reports
- Workplace 2.x MiniApps
- HTTP based web applications

If you just want to integrate one of these application types, please refer to the administration manuals. If the application type you need is not in the list, you can implement your own integrator component.

Implementation Considerations

An application integrator iView creates HTML content that loads the remote application. This is done in three steps:



Connecting to Backend Systems

Loading an application integrator iView

The loading procedure requires that the content included into the portal with the application integrator is accessible from the client computer. The application integrator infrastructure creates the HTML commands to load the remote application. After the application has been loaded, the iView usually stops interaction with the portal.

Deriving iViews from existing portal components is easy:

1. Log in to the portal with a user account that has administrator rights.
2. Select in the top level navigation Content Administration->Portal Content.
3. Use one of the wizards (many of them create AppIntegrator iViews) by selecting a folder with the right mouse key and choosing the option 'Create iView', or:
 - a. Right click a folder of your choice in the Portal Content tree and select New From Portal Archive->iView.
 - b. Search for the com.sap.portal.appintegrator.sap row and select it. Choose Next.
 - c. Select the component of your choice. Choose Next.
 - d. Specify name, id and prefix of your iView. Choose Next.
 - e. On the summary page, choose Finish.

Your iView is displayed in the Portal Content tree and the detail panel will show the property page of your iView. Every component you create your iView from is designed for one remote application. The properties settings of the iView depend on the remote application.

Security Considerations

For security reasons, the following is recommended:

- Requests from the Application Integrator should not include user IDs and passwords for remote systems that the portal user should not see. Otherwise, malicious but otherwise legitimate users with an HTTP sniffer could determine the user IDs and passwords to which they are mapped.

For example, do not map all portal users to a single administrator or super user.
- Use SSL and HTTP POST for communication between systems.

3.8.1.1 Creating an Application Integrator Component

How an Application Integrator Component Works

An application integrator component is a subtype of a portal component. On request to an iView derived from this component, the component implementation reads the property 'TopLayer' from its profile. The property value points to a layer descriptor file. This layer descriptor file describes a single layer of the component. A layer is a Java subclass of class `AbstractIntegrationLayer` that evaluates the portal request object and parameters. While the layer is processed, it changes and/or creates parameters. After the layer is processed, it looks for the next layer descriptor file and if there is one, processes it. When all layers are processed, every layer is called to send the content to the client. Usually only the last layer of the processing chain will create content.

Passing Parameters between Layers

Every layer implementation retrieves its parameters from a map-like structure and passes its calculated parameters back to the same map instance again. Before a layer is processed, the map is filled with parameters from its layer descriptor file. The map is accessible from the layer implementation with methods from the layer base class

`AbstractIntegrationLayer`.

Parameter Sources and Priority

The following list shows the parameters available for the layer implementation, sorted by priority.

1. Parameters set by the current layer.
2. The layer's layer descriptor file.
3. Parameters set by the preceding layer.
4. Parameters contained in the portal component request object (hence, also HTTP request parameters are available).
5. Parameters contained in the portal node object.
6. Parameters contained in the portal component profile.

If none of these parameter sources has a value, the parameter `parameterName.default` is searched in the sources. The order shown above is maintained so you can set a default value in the layer descriptor file without overwriting the value from a source with a lower priority.

End of Layer Processing

The value in the map stored under the name 'NextLayer' is the name of the next layer property file. The processing chain ends when the parameter 'NextLayer' has the value 'NoLayer' (case-insensitive) or is undefined.

Layer Descriptor File

The layer descriptor file has the Java properties format and must contain the following properties:

- **ClassName**
The name of the layer implementation class.
- **OptionalParameters**
Optional parameters for the layer implementation, separated by a comma.
- **MandatoryParameters**
Mandatory parameters for the layer implementation, separated by a comma.

Creating the Component

An application integrator component is a portal component that has following parts:

- The portal component archive descriptor file `PORTAL-INF/portalapp.xml` with a `SharingReference` entry to application `com.sap.portal.appintegrator`.

Connecting to Backend Systems

- A 'starter' class that inherits from `com.sapportals.portal.appintegrator.AbstractIntegratorComponent`. The starter class will be specified as the component class in the `portalapp.xml` file. The implementation of the class will be empty.
- Each layer used in your component needs a corresponding `PORTAL-INF/property/layername.properties` file. Each property file provides properties for one layer. The file content must comply to the Java property file syntax.

Implementing Your Own Layer

You extend the existing layers by implementing the Java class `com.sapportals.portal.appintegrator.layer.IIntegrationLayer` or `AbstractIntegrationLayer`. This `IIntegrationLayer` interface has plenty of methods to implement. The class `AbstractIntegrationLayer` provides the basic functionality needed to implement a layer and is therefore more convenient to use.

AbstractIntegrationLayer Methods

The `AbstractIntegrationLayer` class provides the following methods:

init(..)

This method initializes the layer class to handle a single request. You do not have to override this method. The `AbstractIntegrationLayer` class implementation stores the parameters in object (member) attributes. These attributes are used in other implementations of `AbstractIntegrationLayer`, therefore your layer implementation class should call `super.init(..)` once, in case you override this method.

getNextLayer()

With this method you return the name of the properties file that describes the layer that should be processed next. Your implementation can decide the next layer at runtime but you can also put a 'NextLayer' property into the property file of the layer that specifies the next layer and not override this method.

render(IPortalComponentResponse)

This method generates the output. The `AbstractIntegrationLayer` implementation of this method generates also HTML-comments if the property 'WriteComments' in the layer property file is set to `true`.

The following methods should not be overwritten:

- `epilog`
- `prolog()`
- `processLayer()`

Connecting to Backend Systems

The methods in your layer implementation class are called in the following sequence:

1. Constructor with no argument of your layer class
2. `init(..)`
3. `prolog()`
4. `processLayer()`
5. `epilog()`
6. `getNextLayer()`
7. `render(IPortalComponentResponse)`

When the processing chain contains more than one layer, the calling sequence is as follows:

1. Constructor with no argument of first layer
2. `init(..)` of first layer
3. `processLayer()` of first layer
4. `getNextLayer()` of first layer
5. Constructor with no argument of the next layer
6. `init(..)` of the next layer
7. `processLayer()` of the next layer
8. `getNextLayer()` of the next layer
9. `render(IPortalComponentResponse)` of first layer
10. `render(IPortalComponentResponse)` of the next layer

During the first processing step each layer is initialized, processed and checked if there is a next layer. Rendering is done in a second processing step where every layer in the processing chain has the opportunity to output something. However, it is good practice to produce output only from the final layer in the processing chain.

3.8.1.2 Component `com.sap.portal.appintegrator.sap.Generic`

With the `com.sap.portal.appintegrator.sap.Generic` component any HTTPbased Web application can be integrated very flexible. For example, parameters can be exchanged and SSO can be performed.



As already mentioned before, the Web application is called directly by the client (isolation level URL). Since the portal does not serve as content proxy, the Web application has to be accessible from the client computer, this means that the firewall has to grant a direct connection from the client browser to the application.

iView/Component Parameters

ApplicationParameter

Defines name-value pair parameters that are passed on to the application.

Example: `appParam1=value1&appParam2=value2`

Multiple name-value pairs have to be separated with the ampersand character (&).

DebugMode

In debug mode, setting this parameter to true, returns a screen with debugging output before launching the application. false disables debugging mode.

Id

Defines name-value pair parameters that are passed on to the application. It is used as placeholder and can be useful to define settings for delta links and iView templates.

LAF

Defines the theme (look and feel) subset (for example, .ur (unified rendering), controls).

SSO2Template

Defines a template that is used for SSO parameter `<Authentication>` (if the system defines logon with SAP Logon Tickets). See the Single Sign-on description later in this document for more details.

System

Name of the system alias.

URLTemplate

Defines a template that is processed by the template processor in order to generate the application url. See the Template Processor description later in this document for more details.

.

UserMappingTemplate

Defines a template that is used in SSO parameter `<Authentication>` when the system defines logon with user mapping. See the Template Processor description later in this document for more details.

RequestMethod

Overwrites the default behaviour for retrieving the request method of the redirect (GET or POST). For SSO with user mapping, the default is "POST". "POST" is also used if the parameter list is longer than 1024 characters.

Remark: In EP SP2 Patch 1 this property has to be added manually to the deployment descriptor of the application.

Template Processor

The template processor parses a template (for example, `URLTemplate` for the `com.sap.portal.appintegrator.sap.Generic` component) and replaces the tag expressions that match the context names, with values stored in contexts.

Definitions

Template

A template is a string containing text and tags.

Example: This is a template with a simple `<tag>`.

Tag

A tag is a variable expression followed by an optional modifier sequence in angle brackets.

Example: Enclose tags always in `<angle.brackets>`

Tag expression

A tag expression (or variable expression) is a set of variables separated by dots.

Example: `<tag.expression.with.multiple.variables>`

Context

A context is a data structure that stores name-value pairs, similar to `java.util.Hashtable`. Contexts can be nested.

Variable

A variable is an identifier that matches entries in the context. If a variable name equals a context entry name, the tag is replaced with the value of the context entry.

Example: `<context1.variable1>` is replaced with the value of `variable1` in `context1`.

Modifier sequence

A modifier sequence contains one or more modifiers in square brackets separated by blanks. Modifiers are applied to the substitute in the order of the occurrence.

Example: This is a template with `<modified.tag[URL_ENCODE UPPERCASE]>`

Substitute

A substitute of a tag expression is the result of a processed/substituted tag expression.

Example: `<User.Name>` is the tag expression and Blair Pascal is the substitute.

Use of Templates

Nested contexts are accessed with tag expressions. The variables in the tag expression specify the "path" in the context tree, for example, `<x.y>`. If a variable does not match a context, the tag is replaced by an empty string.

Processed tag expressions can be automatically URLEncoded using the `URL_ENCODE` modifier.

Example:

```
<this.will.be.Urlencoded[URL_ENCODE]>
```

The slash (/) represents the "escape" character in templates. When you put a slash before an angle bracket it is not treated as tag.

Example:

Connecting to Backend Systems

```
Find car: Mileage /< 50000 and Year />
```

Template grammar:

```
template      ::= ( plain | tag ) *
plain         ::= unreserved *
unreserved    ::= ... all characters except the reserved ones
tag           ::= "<" tagexpr ">" | "<" tagexpr ( modif_seq ) ">"
tagexpr       ::= simpletagexpr ( "." simpletagexpr ) *
simpletagexpr  ::= alpha alphanum *
modif_seq     ::= ( alphanum+ " " ) +
alpha         ::= "A" | ... | "Z" | "a" | ... | "z" | "_"
alphanum      ::= alpha | "0" | ... | "9"
reserved      ::= "<" | ">"
```

Example:

```
<Protocol>://<HostAddress>:<Port>/some/path?user=<User.Name[UPR_CASE]>
&
url= <HomeServer.url[URL_ENCODE UPPERCASE]>
```

The question mark (?) marks the parameter section in an HTTP/S url.

Resolution of Expressions

We have the following expressions:

- Expressions with Variables

Expressions with variables have the format `<x>`. They are resolved by a custom provider, the request and the iView profile, in this order, in two iterations. The first iteration looks for the variable name, for example, "x", and the second iteration looks for the default variable name, for example, "x.default".

- Expressions Using a Context

Expressions using a context, like `<x.y>`, offer data retrieval from data providers such as the system landscape or user management. With a context, it is possible to access a specific property. This could be necessary to overcome the resolution rules.

Example:

The same property exists in the profile and in the request. In order to access this property from the profile, you have to explicitly access the property with the name `<Profile.property>`. Accessing it with name `<property>` only, would return the property from the request. The resolution of expressions can be performed either indirectly or directly:

- Directly

If the context is known, an expression like `<User.UserID>` is resolved directly.

- Indirectly

An expression like `<System.url>` is resolved indirectly since the name for the system is not known. It has to be retrieved from the iView profile first before the system landscape can be accessed.

Available Variables

The following variables are available:

- All properties according to the rules given in `iViewParameterName`
- User mapping user. Variable: `<MappedUser>`
- User mapping password. Variable: `<MappedPassword>`
- Authentication string. Variable: `<Authentication>`
- User ID used for accessing applications. Variable: `<LogonUserId>`

This variable contains the portal user ID if no user mapping is maintained. If user mapping exists the variable contains `<MappedUser>`.

Available Contexts

The following contexts are available:

- Look and Feel Context (LAF)
This context provides the following information:
 - Current theme. Variable: `<LAF.Theme>`
 - Base URL of current theme. Variable: `<LAF.BaseUrl>`
 - Full qualified stylesheet url. Variable: `<LAF.StylesheetUrl>`
- Portal Component Profile Context (Profile)
This context provides information from the iView profile.
 - Property name from the iView profile. Variable: `<Profile.[id]>`
- Portal Context (Portal)
This context provides the following portal information:
 - Portal Runtime version. Variable: `<Portal.Version>`
 - Root url of portal: `<Portal.RootComponent>`
- Request Context (Request)
This context provides the following information from the servlet request:
 - Language. Variable: `<Request.Language>`
 - Locale. Variable: `<Request.Locale>`
 - Protocol. Variable: `<Request.Protocol>`
 - Server name. Variable: `<Request.Server>`
 - Port. Variable: `<Request.Port>`
 - Server plus port. Variable: `<Request.ServerPort>`
 - SSO2 ticket. Variable: `<Request.SSO2Ticket>`
 - Any request parameter. Variable: `<Request.[ParameterName]>`

Connecting to Backend Systems

- System Landscape Context (System)

This context provides all the properties of the system landscape. The possible variable names are defined by the system landscape itself:

- System property value. Variable: `<System.[SystemPropertyName]>`

- User Context (User)

This context provides the following user information:

- Accessibility level. Variable: `<User.Accessibility>`
- Logon ID. Variable: `<User.LogonUid>`
- Logon ID. Variable: `<User.UserID>`
- User attribute. Variable: `<User.[AttributeName]>`

Available Modifiers

Following modifiers are available:

- BASE64
Performs base64 encoding.
- HTML_ESCAPE
Escapes the result so that it can be put in HTML documents.
- LOWERCASE
Converts all characters to lower case.
- MD5
Performs MD5 encoding.
- SAP_BOOL
Converts boolean values to SAP notation (true/1 -> 'X', false/0 -> '').
- SAP_ITS_NAMESPACE
Replaces '/' with '-' (used for IACs)
- TRIM
Removes all leading and trailing whitespaces.
- UPPERCASE
Converts all characters to upper case.
- URL_DECODE
Performs url decoding
- URL_ENCODE
Performs url encoding
- PROCESS_RECURSIVE
Forces recursive resolution of templates. If the result of a template is also a template, another recursive step of template processing is performed.

Single Sign-On

Component `com.sap.portal.appintegrator.sap.Generic` allows you to perform a Single Sign-On to HTTP Web applications by including the tag `<Authentication>` in the URL template. Prerequisite is a system with a system definition that includes the following properties:

- User mapping type, if user mapping is used.
- Logon method

The tag `<Authentication>` returns the result of the `<UserMappingTemplate>` or the `<SSO2Template>`.

`UserMappingTemplate:`

For user mapping you can set the `UserMappingTemplate` as follows:

- Basic authentication

```
<MappedUser>:<MappedPassword>@
```

The `URLTemplate` would then look like this:

```
http://<Authentication>protectedserver.com/xyz
```

The parameters `user` and `password` are application-specific. They need to be retrieved from, for example, the HTML source code of the logon form. The `URLTemplate` would then look like this:

```
http://protectedserver.com/xyz?<Authentication>
```

- HTTP form-based login

```
user=<MappedUser>&password=<MappedPassword>.
```

`SSO2Template:`

If you use SAP logon tickets for authentication, you can specify an authentication template in parameter `SSO2Template` like:

```
MYSAPSSO2=<Request.SSO2Ticket>
```

3.8.2 Connector Framework

Purpose

The connector framework provides the infrastructure to create platform independent connectors. Connectors are connectivity providers for Web applications. The framework provides the information that is necessary for the connector to gain access to its data source, the Enterprise Information System (EIS). The connectors comply with the Java 2 Platform Enterprise Edition (J2EE) standard and its J2EE Connector Architecture (JCA 1.0) specifications. The JCA 1.0 refers to connectors also as resource adapters.

Use

The Portal Runtime (PRT) runs as a servlet on the Web application server (WAS). The PRT supplies different services to SAP Enterprise Portal users, such as user management and security. These and other services of the PRT are provided to portal applications, which use connectors deployed on the application server (J2EE Engine) to communicate with a back-end application (EIS). The portal applications, also referred to here as portal content, can be created automatically. The wizard framework is a PRT service for creating such applications automatically. The wizard framework requires system connectivity parameters. These parameters are provided by a system template, which is supplied together with each connector for its respective EIS. However, portal content may also be developed with code, using any of the public portal APIs in general, and the Connector Framework API in particular. These APIs can be used for connecting the content that populates the portal to back-end systems, for creating new connectors for additional systems.

The connector framework is a J2EE service that offers advantages when connectors are implemented and used in a managed environment, such as JNDI lookup and connection pooling. However, working in [non-managed mode \[Page 33\]](#) is also possible. The implementation and deployment of a connector is affected accordingly. See [Connector Deployment \[Page 33\]](#) for more details.

Connector Framework as a Web Service

The connector framework is also available as a Web service, using the SOAP protocol, to transfer requests and responses between applications, such as portal applications and specific connectors.

Integration

The connector framework consists of the connector framework API, the connector wizard for new connectors (to conveniently create a connector project in NetWeaver Development Studio), the connector Web service, and documentation, including examples.

The connector framework extends the JCA 1.0 standard, to meet the requirements of SAP Enterprise Portal users.

The documentation contains the developer guide, which describes the concepts of the connector framework structure, management, and functionality, as well as installation information, development environment prerequisites, and information about the deployment of connectors created with the generic API.

The reference guide uses standard Javadoc, supplying the detailed technical information necessary to implement the generic connector API. For detailed reference information about the interfaces mentioned here, see the Javadocs for the portal and portal runtime APIs, located on the SAP Developer Network (SDN) at www.sdn.sap.com/irj/sdn/javadocs.

3.8.2.1 J2EE Connector Architecture (JCA)

The J2EE Connector Architecture (JCA) is a specification that defines the standard architecture for connecting the Java 2 Platform, Enterprise Edition (J2EE) platform to heterogeneous Enterprise Information Systems (EIS's), which may include, for example, ERP and database systems. The mechanisms that the connector architecture defines are scalable and secure and enable integration of the EIS with application servers and enterprise applications.

Connecting to Backend Systems

Standard connectors, called *resource adapters* in the JCA specification, can be supplied by any given EIS. The connectors are software drivers used by an application to connect to an EIS. The connectors can be plugged into an application server, such as *SAP Web AS Java*, and provide connectivity between an EIS, the application server, and the enterprise application.

Once an application server supports the connector architecture it provides seamless connectivity to multiple EIS's. Likewise, an EIS vendor provides one standard resource adapter and it has the capability to plug into any application server that supports the connector architecture. JCA 1.0 defines standard Java interfaces for simplifying the integration of enterprise applications with J2EE-based Java applications. The connector is a component library that can be used in Java from the developer.



The Connector Framework supports the latest version of JCA, although its implementations and extensions are based on JCA 1.0.

JCA 1.0 Specification

The Connector Framework extends the J2EE Connector Architecture, so you should be familiar with the specification to understand the implementation of connectors, their use and the environment in which the connector works.

The JCA 1.0 specification document deals with the various aspects of the standard connector architecture:

- Standard client API, as defined by the Common Client Interface, for enabling interactions between application components and heterogeneous EIS.
- Defines the requirements that architecture-compliant connectors, together with the application server, support the system contracts. Detailed requirements are specified for every system.
- Discusses the connector architecture for roles. Scenarios are included to illustrate the concepts.
- Specifies the responsibilities on system level for connection management between an application server and a connector. The description also explains deployment and illustrates scenarios of the connection management.
- The specification relates how to create a connection in a non-managed environment, for example, Java applications and applets.
- Specifies the transaction management on system level between an application server (and supported transaction manager) and an EIS.
- Discusses security architectures for the integration of EIS with the J2EE platform, adding EIS-specific security details to the security requirements specified in other J2EE specifications.
- Describes the security contract between the application server and the EIS, specifying the responsibilities of the connector and the application server in support of the security contract.
- Specifies requirements for packaging and deploying a resource adapter.
- Specifies the Java API that a J2EE-compliant application server (and its containers) must provide for a connector at runtime.
- Describes standard exceptions for errors that may occur in the connector architecture.

You can download the Sun Microsystems J2EE Connector architecture specification at:
java.sun.com/j2ee/download.html#connectorspec.

Advantages of a Generic Connector API

A standard provides the guidelines to develop connectors for different applications and different connectors for the same applications, with the same language and technology.

Main advantages are:

- Depending on the EIS, the connectors support standard subsets of interfaces specified in the connector framework. Guidelines exist for the service types provided by connectors.
- Third parties can independently develop the application-specific connectors with the connector framework.
- The connector is platform independent.
- The connector (including related services) is the only access layer for the application.
- The connector API is provided as a Web service. This allows http access to the EIS using the SOAP protocol.
- The Java code can be generic and therefore be developed only once for any EIS.
- Integration of projects is possible.
- Reduced training and maintenance costs.

Using the Connector Architecture

All applications that need to access data from the EIS are users of the connector service (for example, iViews in the Enterprise Portal).

The user needs to do the following:

- establish a connection to the EIS
- retrieve connector metadata and its supported capabilities
- data access (query execution and function execution) to a specific EIS
- get EIS metadata and business models
- call EIS-specific methods (for example, running SAP transactions)

3.8.2.2 Connector Framework Structure

The connector framework consists of interfaces described by the JCA 1.0 specification and additional interfaces, defined by the connector framework, which extend the JCA. The connector framework extensions are the focus of this document.

The information described in this section is intended both for the developers of connectors and for the developers of applications that consume existing connectors already deployed on the WebAS.

In addition to the connector metadata and capabilities information, see also:

[Connection Interfaces \[External\]](#)

[Metadata Retrieval \[Page 33\]](#)

[Interface IStructure \[Page 33\]](#)

[Data Access and Method Execution \[Page 33\]](#)

[Functions and Queries \[External\]](#)

Connector Metadata

The connector metadata contains basic JCA 1.0 parameters and connector framework extension parameters.

JCA parameters are:

- EIS version
- Vendor name
- Description

Connector framework parameters are:

- Default Date and Time Format
- Maximum number of connections allowed
- Default timeout for the connection and queries

Capabilities Manifest

The connector framework `ConnectorMetadata` interface uses the method `getCapabilities` to retrieve an `ICapabilities` object. The `ICapabilities` interface returns the capabilities of a specific connector. The basic connector has a generic interface to find out what the connector is capable of doing, such as multi-language support, query cancel support, or returning results in chunks of data for higher performance. A specific connector may support only some of the capabilities supported by the connector framework; it can also extend the list of capabilities.

The connector framework has an `ICapabilities` interface that reflects the capabilities of a connector. The interface `ICapabilities` defines the connector capabilities. `ICapabilities` has two methods by which it is possible to retrieve all capabilities supported with the current connector (`getAllSupportedCapabilities`), and whether or not a specific capability is supported (`supports`).

A connector has to implement the `ICapabilities` interface for a specific capability. The method (`supports`) will return true to indicate that the connector supports a specific capability.

For more details, refer to the Javadocs for the portal and portal runtime APIs, which are located on the SAP Developer Network (SDN) at www.sdn.sap.com/irj/sdn/javadocs.

3.8.2.2.1 Interface ICapabilities

The interface `ICapabilities` defines the connector capabilities. `ICapabilities` has two methods by which it is possible to retrieve all capabilities supported with the current connector (`getAllSupportedCapabilities`), and whether or not a specific capability is supported (`supports`).

Connector Framework: Supported Capabilities

Connector Capability	Description of Supported Capability
CONNECTION_METADATA_PROPERTY_GROUPS	Retrieval of connection property groups other than the default
DELETE	DELETE operations through queries

Connecting to Backend Systems

INSERT	INSERT operations through queries
INTERACTION_EXECUTION	JCA Interaction interfaces, meaning support for EIS function execution
INTERACTION_EXECUTION_OUTPUT_INDEXED	Output format of the Interaction.execute() function as an indexed record
INTERACTION_EXECUTION_OUTPUT_MAPPED	Output format of the Interaction.execute() function is a mapped record
METADATA_FUNCTIONS	Retrieval of function metadata from EIS
METADATA_FUNCTIONS_REG_EXP	Retrieval of function metadata from EIS using regular expressions; for example, the connector implements - Set getFunctions(String functionNameRegularExpression, String functionGroupRegularExpression)
METADATA_OPERATORS_EQUAL	Retrieval of conditions with the equal operator - Equal conditions are conditions of type: A.x = value object A attribute x is equal to value
METADATA_OPERATORS_GREATER_THAN	Retrieval of conditions with the greater than operator - Greater than conditions are conditions of type: A.x GREATERTHAN value object A attribute x is greater than value
METADATA_OPERATORS_GREATER_THAN_OR_EQUAL	Retrieval of conditions with the greater than or equal operator - Greater than or equal conditions are conditions of type: A.x GREATERTHANOREQUAL value object A attribute x is greater than or equal to value
METADATA_OPERATORS_LESS_THAN	Retrieval of conditions with the less than operator - Less than conditions are conditions of type: A.x LESSTHAN value object A attribute x is less than to value
METADATA_OPERATORS_LESS_THAN_OR_EQUAL	Retrieval of conditions with the less than or equal operator Less than or equal conditions are conditions of type: A.x LESSTHANOREQUAL value - object A attribute x is less than or equal to value
METADATA_OPERATORS_LIKE	Retrieval of conditions with the like operator - Like conditions are conditions of type: A.x LIKE value object A attribute x is like to value
METADATA_OPERATORS_NOT_EQUAL	Retrieval of conditions with the not equal operator - Not equal conditions are conditions of type: A.x NOTEQUAL value object A attribute x is not equal to value
MULTILANGUAGE	Multilanguage - input/output values can be multilanguage
NATIVE	INative interface,

Connecting to Backend Systems

QUERY	Queries
QUERY_BLACKBOX_JOIN	Execution of queries that contain blackbox relation
QUERY_CANCEL	Cancellation of queries during the execution
QUERY_CRITERIA_LOGICAL_AND	Compound criteria using AND as operator in the WHERE clause
QUERY_CRITERIA_LOGICAL_NOT	Logical criteria using NOT as operator in the WHERE clause
QUERY_CRITERIA_LOGICAL_OR	compound criteria using OR as operator in the WHERE clause
QUERY_CRITERIA_PREDICATE_ATTRIBUTE_EQUALS_CONSTANT	Only predicate criteria of the following types in the WHERE clause: businessObject.attribute = CONSTANT Blackbox relation
QUERY_CRITERIA_PREDICATE_ISNULL	IS NULL predicate criteria in the WHERE clause
QUERY_CRITERIA_PREDICATE_SET	IN () predicate criteria in the WHERE clause
QUERY_FROM_ALIAS	Aliases for business objects in the FROM clause of the query statement
QUERY_FROM_INNERJOIN	inner joins in the FROM clause of the query statement
QUERY_FULLSCAN	Full-scan query – where full scan is not supported, only query statements where the field attributes participating in the WHERE clause are indexes will be supported (WHERE clause is mandatory)
QUERY_NATIVE	INativeQuery interface - can execute native SQL queries
QUERY_RESULT_CHUNK_SUPPORT	Retrieval of query resultSets in chunks - where the size of a chunk is defined by the number of records returned— retrieval chunks is done consecutively, one by one
QUERY_SELECT_AGGREGATION_AVG	AVG (attribute) or AVG (DISTINCT attribute) aggregation function in the SELECT clause
QUERY_SELECT_AGGREGATION_COUNT	COUNT (attribute) or COUNT (DISTINCT attribute) aggregation function in the SELECT clause
QUERY_SELECT_AGGREGATION_COUNT_STAR	COUNT (*) or COUNT (DISTINCT *) aggregation function in the SELECT clause
QUERY_SELECT_AGGREGATION_DISTINCT	DISTINCT keyword in the aggregation function in the SELECT clause
QUERY_SELECT_AGGREGATION_MAX	MAX (attribute) or MAX (DISTINCT attribute) aggregation function in the SELECT clause

Connecting to Backend Systems

QUERY_SELECT_AGGREGATION_MIN	MIN (attribute) or MIN (DISTINCT attribute) aggregation function in the SELECT clause
QUERY_SELECT_AGGREGATION_SUM	SUM (attribute) or SUM (DISTINCT attribute) aggregation function in the SELECT clause
QUERY_SELECT_ALIAS	aliases in the SELECT clause
QUERY_SELECT_DISTINCT	DISTINCT keyword in the SELECT clause
QUERY_SELECT_LEAF_BUSINESS_OBJECT_ATTRIBUTE	Query statement where the FROM clause is a join tree—primary-foreign key joins
QUERY_SELECT_SINGLE_BUSINESS_OBJECT_ATTRIBUTE	Queries where the SELECT clause contains attributes from a single business object
QUERY_SORT	ORDER BY clause in the query statement
TRANSACTIONS	Transactions
UPDATE	UPDATE operations through queries

3.8.2.2.2 Interface IStructure

The interface `IStructure` defines the data type of a parameter or attribute. This interface extends the JCA 1.0 standard by defining additional complex data types as well as the primitive data structure, like, *string*, *integer*, *byte*, *Boolean*, *date* and so on. The additional data types and their descriptions are:

Data type	Description
Recordset	Array of records
Array	Array of fields which are of the same data type.
Record	Array of fields each of which is a member of a different column
Field	A data structure having only a primitive type.

Data Output Structure	Description
For queries	It is defined by interface <code>IRecordSet</code> and is a subset of <code>java.sql.resultset</code> of the current chunk. It is the part of the total record set, defined by a minimum and maximum number of records, returned at one time.
For functions	What is expected in the output parameters of the specific function.

Connecting to Backend Systems

3.8.2.2.3 Interface ConnectionSpec

The primary task of the connector framework is to create the basic connection. The `ConnectionSpec` interface is a collection of properties that have to be specified to connect to a back-end application.

A connection property consists of:

- System-parameter or user-parameter
The system-parameter is shared among all users, while a default value for the user-parameter may be overridden by the user mapping.
- Is the parameter mandatory
- The data type of the parameter
- The default value of the parameter
- A list of valid values for the parameter

The `ConnectionFactory` gets an instance of the connection based on the given `ConnectionSpec` collection.

The `Connection` interface is the entry point for the connector to perform an operation like getting metadata, launching a query, or getting a native interface. Once the basic connection has been established, additional functions can be enabled.

3.8.2.2.4 Metadata Retrieval

The metadata retrieval extends the JCA 1.0 standard. It may include functions, objects, and/or relations.

Metadata type	Description
Business Objects Group	Namespace entities that have nested groups or objects.
Business Object	Represents the business object of the back-end application. It is a combination of methods, attributes and relations.
Attributes	Composite elements of business objects of a certain type. It can also provide information on indices to prevent time consuming queries.
Methods	Methods are independent of any business object or group. It is a list of all functions existing in the application. Operative elements of objects and groups behave in the same manner as predefined application functions.

Connecting to Backend Systems

Relations	<p>Relations are elements describing the connections between different entities in an application.</p> <p>Relations are possible between business objects, attributes, or a group of attributes.</p> <p>All relations are one-directional; a bi-directional relation is a particular case in which one-directional relations between the entities already exist.</p> <p>Relations are presented to the user of the connector as string object.</p> <p>There are following relation types:</p> <ul style="list-style-type: none"> • Not editable The application does not permit to add or extend the relation. • Editable - Default Implementation. Obj1.AttrX = Obj2.AttrY • Editable - Custom Implementation When a relation is represented by a function call or some other connector-specific logic.
-----------	---

3.8.2.2.5 Data Access and Method Execution

In addition to metadata retrieval, the connector framework provides additional features for data access and method execution.

The connector allows data access with following functions:

Feature	Description	Provided by
Native queries (<code>INativeQuery</code>)	The connector accepts commands and queries stated in the application language, for example, SQL and MDX. The returned data type is <code>Java.lang.Object</code> and the connector passes it on to the user application <i>as it is</i> , without parsing or interpretation.	Connector Framework
Function or interaction (<code>IInteraction</code>)	The connector invokes a given function, that exists in the application as follows: Every parameter supports data type, owner, valid values, in/out direction, default value (if optional), and so on. See section Interface ConnectionSpec [Page 33] for more details.	JCA
Native API Access (<code>INative</code>)	The <i>native module</i> wraps the interface visible to the application. The connector returns a reference to a requested native interface and from this point on, the user of the connector interacts with the EIS directly through that interface.	Connector Framework

Connecting to Backend Systems

	<p>The Java Connector client (JCO) service is an example of this. The JCO client service implements the connectivity to SAP back-end systems. The JCO client service gets an SAP back-end system connection from of the connector framework first and then calls for the JCO interface to get a native function.</p> <p>An example of this for a relational database management system would be</p> <pre>java.sql.Connection.</pre>	
--	---	--

3.8.2.3 Using Existing Connectors

Whether they come with the installation of SAP Enterprise Portal or are developed separately and installed for portal users, the connectors to back-end systems are deployed on the J2EE application server level. This offers distinct advantages for connector consumers in a managed environment, which is typically the case for NetWeaver users. Among the primary advantages are:

- JNDI lookup

Instead of having to create a connection factory instance, the content developer need only implement the connector gateway service of the portal. This is exemplified by the samples in the section about the [JDBC Connector \[Page 33\]](#).

- Connection pooling

The application server manages connections, reusing already open ones for new requests, instead of the performance-reducing opening and closing of connections for each request for an EIS connection.

The main task of the developer of portal content, such as iViews that retrieve information from a back-end system (EIS), is to open a connection to that system.

The SAP NetWeaver portal comes with the following ready-to-use connectors. See the samples for help with using them:

- a [JDBC Connector \[Page 33\]](#) for universal database connectivity
- an [SAP System connector \[Page 33\]](#) for connectivity to back-end SAP systems
- a [Web Service connector \[Page 33\]](#) for interacting with any Web service

Prerequisites

Following are the prerequisites for using connectors that are deployed in the portal.

File/Environment	Description	Source
GenericConnector.jar	Required to test the application	Comes with <i>SAP Enterprise Portal</i>
ExtendedConnector.jar	Adds functionality to generic connector	Comes with J2EE application server.
Connector.jar	Contains the JCA 1.0 API	Comes with J2EE application server
JAAS.jar	Required by the connector framework.	Comes with J2EE

Connecting to Backend Systems

(Java Authentication and Authorization Service)		application server
JTA.jar (Java Transaction API)	Required by the Connector Framework.	Comes with J2EE application server
EIS (Enterprise Information System)	Required to test the application.	Has to be installed separately
Java Application Server	Any J2EE JCA 1.0-compliant WAS to work with servlets without the <i>SAP Enterprise Portal</i> . Recommendation: <i>SAP Web AS 6.40 Java</i>	Has to be installed separately
PRT (Portal Run Time)	Required to test the application in the <i>SAP Enterprise Portal</i> environment	Comes with <i>SAP Enterprise Portal</i>
Portal Wizard Framework	The portal wizard framework is an effective tool to create a portal application that uses a connector.	Comes with <i>SAP Enterprise Portal</i>
Connector Web Service	Necessary to communicate between application and connector as Web service, using http calls and SOAP.	Comes with <i>SAP Enterprise Portal</i>

3.8.2.3.1 Using the SAP System Connector

The *SAP System* adapter for the *SAP Enterprise Portal* 6.0 is a connector framework implementation based on the J2EE Java Connectivity Architecture (JCA 1.0), the SAP RFC library, and the SAP Java Connector (JCO). It enables connectivity to SAP ERP systems, CRM (Customer Relations Management) and BI (Business Intelligence).

The connector framework has *connection management*. This overcomes a common problem that occurs when using the *JCo client service*, where the developer has to take care about connection management himself. Therefore it is strongly recommended not to use *JCo client service* any more and migrate existing EP 5.0 applications that use the *JCo client service* to *Connector Framework*.

The *Connector Framework BAPI Example* in the PDK

See also

[JCO Client Service](#)

[\[External\]Prerequisites for using a connector \[Page 33\]](#)

Connector Gateway Service

Example for a connection to a SAP system using the *connector gateway service*:

```

IConnection connection = null;
try { // get the Connector Gateway Service

```

Connecting to Backend Systems

```

    Object connectorservice = PortalRuntime.getRuntimeResources().get
Service( IConnectorService.KEY);
    IConnectorGatewayService cgService =(IConnectorGatewayService) co
nnectorservice;
    if (cgService == null) {
        response.write("Error in get Connector Gateway Service <br>");
    }
    try {
        connection = cgService.getConnection(sapsystem, request);
    } catch (Exception e) {
        response.write("Connection to SAP system failed <br>");
    }
    if (connection == null) {
        response.write("No connection <br>");
    }
    else {
        response.write("Connection succesful");
    }
} catch (Exception e) {
    response.write("Exception occurred");
}
}

```

Executing a BAPI Function

In the following example we execute a BAPI function. The result is a returned *RecordSet* object. A *RecordSet* object has a pointer that points to its current data row. Initially, the pointer is positioned before the first row. The method `next` moves the pointer to the next row. If there are no more rows, the method `next` returns *false*.

```

try {
    // Get the Interaction interface for executing the command
    IInteraction ix = connection.createInteractionEx();
    // Get interaction spec and set the name of the command to run
    IInteractionSpec ixspec = ix.getInteractionSpec();
    // the well known example BAPI SALESORDER
    String functionName = "BAPI_SALESORDER_GETLIST";
    // Put Function Name into interaction Properties.
    ixspec.setPropertyValue("Name", functionName);
    // return structure
    String function_out = "SALES_ORDERS";
    RecordFactory rf = ix.getRecordFactory();
    MappedRecord input = rf.createMappedRecord("input");
    // put function input parameters
    input.put("CUSTOMER_NUMBER", new String("0000001172"));
    input.put("SALES_ORGANIZATION", new String("1000"));
    MappedRecord output = (MappedRecord) ix.execute(ixspec, input);
    Object rs = null;
    try {
        Object result = output.get(function_out);
        if (result == null) {
            rs = new String(" ");
        } else if (result instanceof IRecordSet) {
            rs = (IRecordSet) result;
        }
        // result object returned
    } else {
        rs = result.toString();
    }
}

```

Connecting to Backend Systems

```

    }
  } catch (Exception ex) {
    printException(ex);
  }
  return rs;
} catch (Exception e) {
  printException(e);
}

```

Connecting to an SAP System on WebAS 6.20 without the Connector Service

You can connect to a SAP system directly by providing the username and the password, for example for testing purposes. The *EISConnection* can be initialized directly with JNDI support.

```

// physical connection
IConnection mm_con = null;
//connection factory of the SAP connector to get the connection
IConnectionFactory connectionFactory;
Context initctx = null;

try {
  //get the initial JNDI context
  Hashtable env = null;
  initctx =
    new com.sapportals.portal.prt.jndisupport.InitialContext(env);
  // perform JNDI lookup to get the connection factory
  connectionFactory =
    (IConnectionFactory) initctx.lookup("EISConnections/SAPFactory");
} catch (Exception e) {
}

try {
  // retrieve the ConnectionSpec and set the values
  IConnectionSpec spec = connectionFactory.getConnectionSpec();

  /*
   * set connection properties according to SAP JCO javadoc:
   * static String[][] login_params = {
   *     *{ "client" , "000" },
   *     *{ "user" , "timtaylor" },
   *     *{ "passwd" , "binford" },
   *     *{ "language", "EN" },
   *     *{ "ashost", "mymachine.mycompany.com"};
   *     *...
   * };
   */
  // set properties
  spec.setPropertyValue("client", "000");
  spec.setPropertyValue("user", "timtaylor");
  spec.setPropertyValue("passwd", "binford");
  spec.setPropertyValue("lang", "EN");
  spec.setPropertyValue("ashost", "myserver.mycompa??/Ad??Yd??Rny.
corp");

```

Connecting to Backend Systems

```
spec.setPropertyValue("sysnr", "00");

// Retrieve the connection handle
mm_con = connectionFactory.getConnectionEx(spec);
mm_con.close();

} catch (Exception e) {
}
```

Connecting to an SAP System on WebAS 6.40 without the Connector Service

The connector JDNI name was changed from WebAS 6.20 to 6.40. In 6.20, as shown in the above sample, look for "EISConnections/<connector_name>" to get a connector factory instance.

When migrating code with such a hard-coded look-up string to WebAS 6.40, the JNDI lookup name for the connector needs to be changed to the following format:

deployedAdapters/<connector_name>/shareable/<connector_name>

For example:

```
...
...
...
try {
    //get the initial JNDI context
    Hashtable env = null;
    initctx =
        new com.sapportals.portal.prt.jndisupport.InitialContext(env);
    // perform JNDI lookup to get the connection factory
    connectionFactory =
        (IConnectionFactory) initctx.lookup("deployedAdapters/SAPFactory
/
    shareable/SAPFactory for SAP connector");
    deployedAdapters/SAPFactory/shareable/SAPFactory for SAP
connector
    ...
    ...
    ...
```



It is highly recommended that content developers use the portal connector gateway service, as described above, to establish a connection to the back end. Code that uses this service doesn't require any change when migrating to WebAS 6.40.

See also:[Using a JCO Client for Several Portal Requests \[External\]](#)[Using Connectors with the Distributed Query Engine \[External\]](#)[Using the JDBC Connector \[Page 33\]](#)**3.8.2.3.2 Using the JDBC Connector**

The JDBC adapter for the SAP Enterprise Portal 6.0 is a connector framework implementation based on JDBC.

There are two basic ways to get a connection using the JDBC Connector:

- Via the Connector Service—this approach being faster, simpler, and benefits from the portal's user mapping from the system
- Direct connection using the J2EE JNDI

Connecting via the Connector Service

Example for a connection to a JDBC system using the *connector service*:

```
IConnection connection = null;
// get the Connector Gateway Service
String jdbcSystem = "pubs";
IConnectorGatewayService cgService = (IConnectorGatewayService)

PortalRuntime.getRuntimeResources().getService(IConnectorService.KEY)
;
ConnectionProperties cp = new
ConnectionProperties(portalRequest.getLocale(),
    portalRequest.getUser());
try {
    connection = cgService.getConnection(jdbcSystem, cp);
} catch (Exception e) {
} finally {
    try {
        if (connection != null) {
            connection.close();
        }
    } catch (ResourceException re) {
    }
}
}
```

Connecting via J2EE

Example for a connection to a JDBC system using the J2EE JNDI:

Connecting to Backend Systems

```

// physical connection
IConnection connection = null;
//connection factory of the JDBC connector to get the connection
IConnectionFactory connectionFactory;
Context initctx = null;
try {
    //get the initial JNDI context
    Hashtable env = new Hashtable();

    env.put("domain", "true");
    initctx = new com.sapportals.portal.prt.jndisupport.InitialContext(env);
    // perform JNDI lookup to get the connection factory
    connectionFactory = (IConnectionFactory)

initctx.lookup("deployedAdapters/JDBCFactory/shareable/JDBCFactory");
    // retrieve the ConnectionSpec and set the values
    IConnectionSpec spec = connectionFactory.getConnectionSpec();
    // set properties
    spec.setPropertyValue("UserName", "sa");
    spec.setPropertyValue("Password", "admin");
    spec.setPropertyValue("driver",
"com.sap.portal.jdbc.sqlserver.SQLServerDriver");
    spec.setPropertyValue("url", "jdbc:sap:sqlserver://localhost:1433;
DatabaseName=Northwind");
    // Retrieve the connection handler
    connection = connectionFactory.getConnectionEx(spec);
} catch (Exception e) {
} finally {
    try {
        if (connection != null) {
            connection.close();
        }
    } catch (ResourceException re) {
    }
}
}

```

See also

[Prerequisites for using a connector \[Page 33\]](#)

3.8.2.3.3 Using the Web Service Connector

The Web Service Connector, supplied by SAP with EP 6.0 on WebAS 6.40, is an implementation of the NetWeaver portal connector framework API. It provides the ability to interact with any Web services, regardless of source, with the following advantages over connecting to Web services by other means:

Connecting to Backend Systems

- As part of the portal connector framework, it uses the UME (user management engine) for authentication and user mapping.
- Enables the use of automatic tools (such as the portal iView Wizard and Visual Colmposer) for the creation of iViews.
- No offline proxy generation and deployment are required.

Technologies Used

- WSDL (Web Service Definition Language)
- SOAP (Simple Object Access Protocol)
- SAAJ (SOAP with Attachments for Java) – integrated with J2EE Engine

Connectivity

The only connection property that is mandatory for using the Web Service Connector is the WSDL URL for the specific Web service.

Authentication

The Web Service Connector supports basic authentication, SSO, and no authentication.

No Proxy Client Generation

Using the connector to consume Web services does not involve the use of any Web service client automatically created and deployed on the user machine. The need to create classes to

- create SOAP requests
- sending them
- receiving responses
- parsing the responses

is made unnecessary because the Web Service Connector implements the portal connector framework API, and because it uses the JAAS standard. It can supply input and output variables on-the-fly without the need of a proxy client generation.

Supported Capabilities

The following are supported by the Web Service Connector:

WSDL-SOAP Binding Parameters

- All valid value combinations of the WSDL *style* and *use* parameters, which indicate how to translate a WSDL binding to a SOAP message.
 - *Style* values are *RPC* and *document*.
 - *Use* values are *encoded* and *literal*.

The binding possibilities are:

- RPC/encoded
- RPC/literal
- Document/encoded (Not in use: It is legal WSDL, but this style is not WS-I compliant.)

Connecting to Backend Systems

- Document/literal
- Inclusion of other WSDLs in the WSDL of the target Web service (for example: `<import location = <some_URL>>`).
- If the Web service WSDL is of the *document* style, the Web Service Connector will support an associated schema, and also tags imported from other schemas (for example: `<import namespace = <some_URL>>`).

J2EE Dynamic Proxy

The Web Service Connector supports the use of the J2EE dynamic proxy (a component of the J2EE Web Services Security service). The dynamic proxy URL property can be defined to a WSIL URL, in addition to a WSDL URL, thereby extending the capabilities of the Web Service Connector.

The dynamic proxy can also be configured to an SAP Web service.

Creating and configuring the dynamic proxy in the J2EE Visual Administrator:

1. Launch the Visual Administrator and select the service Web Services Security.
2. Expand *Web Service Clients* → *sap.com* and select *DynamicWSProxies*.
3. Choose *Create* and set a name for the destination.
4. Configure the proxy parameters under the Transport Security tab.
 - For configuring the proxy to a WSIL URL, just enter the desired URL in the URL field.
`http://myserver:50000/inspection.wsil`
 If any authentication is required, choose the authentication type in the Logon Data section and enter the appropriate values.
 - For configuring to an SAP system WSIL, for example for ESA (Enterprise Service Architecture), enter the URL as follows:
`http://<server>:<port>` where the `<server>` is the R/3 server and the port is 50000 + `<SAP application instance number>`
 Enter the appropriate values for the System ID, Client, and Language parameters and then configure authentication in the Logon Data section.

For more information, see [Web Services Security \[External\]](#) and [Security Configuration \[External\]](#).

Constraints

The following are not supported by the Web Service Connector:

- Ignores SOAP headers, whether in request or response.
- Fault messages are treated as CF RuntimeException with the fault message text as the Exception message
- SOAP attachments are not supported whether in the request or response
- If the Web service WSDL is of the style *document/literal*, schema elements must follow these rules to work with the connector framework:
 - Does not use unsupported tags: `<any>`, `<choice>`, `<union>`, `<group>`, `<attributeGroup>`

Connecting to Backend Systems

- Schema must not contain recursive element definitions (an element may not include itself in its definition)

Data Handling

The data returned by means of the Web Service Connector can be rendered as viewable content in the NetWeaver portal either by means of Java code written by the content developer, or by automatic tools, such as the iView wizard. The automatic tools create Web Service iViews based on an iView template provided with the portal and uses iView wizard runtime code, which handles all function iViews for all connector framework connectors. (For more information about the system template, see the [System Landscape \[External\]](#) section of the Portal Administration Guide.)

If content is to be provided by means of the automatic tools, it is important to be aware of how complex data types are handled by the Web Service Connector. For detailed information about this, see [Data Type Handling by the Web Service Connector \[Page 33\]](#).

3.8.2.3.3.1 Data Type Handling by the Web Service Connector

How the Web Service Connector handles the various possible data types sent to/returned from a Web service depends on the means by which the data is to be rendered in the portal:

- by writing Java code, in which case the developer decides how to implement the display of the data returned
- by means of automatic tools, such as the iView wizard

In either case, attention must be first turned to the properties of the Web Service system template within the system administration framework of the portal.

Mapping WSDL Schema Elements to Connector Framework Data Types

The following table shows how the Web Service Connector maps WSDL schema elements to connector framework data types.

WSDL to Connector Framework Data Type Mapping

XSD Schema Element	Connector Framework Datatype	Sample/Remarks
<xsd:element> with xsd primitive type attribute value	Field	<xsd:element name="Name" type="xsd:string"/>
<xsd:all>	Record	
<xsd:element> with 1 or more attributes <xsd:attribute>	Record	Where the first field of the record represents the element and all the element attributes are the following record fields
<xsd:sequence>	Recordset	

Portal Properties of the Web Service System

The Web Service system in the portal has, under category Web Service, three properties:

Connecting to Backend Systems

Web Service System Properties

Parameter	Values	Description
Force SAP UI Parameter Management	Y/N Default value: Y	Declares whether or not the Web Service Connector will rearrange input/output parameters of the specific Web service in order to adhere to the SAP UI Parameters standard.
WSDL Caching Days	none/1/30/infinite Default value: 1 (day)	Caching period of the WSDL file
WSDL URL	A valid URL	The URL of the Web service WSDL

Forcing SAP UI Parameter Management

Regarding how the connector deals with the data types of Web service input/output parameters, it is this system property that concerns the content developer.

Value ‘N’ Chosen

If this flag is off (the value selected is *N*), the Web Service Connector does nothing while processing the Web service WSDL file. The connector framework supports all input/output data structures:

- field – of primitive type (for example, int, String, double, etc.)
- array – such as a column in a table (not currently supported by the Web Service Connector)
- record – a set of fields
- recordset – collection of records (table structure)

When one of the fields of a record, or one of the columns of recordset contains data of type record, or recordset, automatic UI tools will not be able to create iViews displaying parameters returned by the Web Service Connector, if their data structure is non-compliant or “nearly flat” (explanation below). The developer can use the connector framework API as desired. (See the code sample using the Google Web service.)

Value ‘Y’ Chosen

The Web Service Connector is capable of passing all data returned to automatic SAP UI tools. However, it is not capable of passing nested data as nested data. The connector is only able to pass data that is “flat,” or “nearly flat.”

If the *Force SAP UI Parameter Management* flag is on (*Y* is selected), the Web Service Connector will manipulate the metadata it will parse from the WS WSDL so that it will appear as flat. The next section describes the supported data structures.

SAP UI-Supported Data Structures

The data structure of input/output parameters that the Web Service Connector supports, for consumption by SAP UI tools, are flat or “nearly flat.” This means that an input/output parameter may be either:

1. a primitive data type (byte, short, int, long, float, double, char, boolean)

Connecting to Backend Systems

We call this a **flat** data structure.

2. a record whose fields:
 - contain data of primitive types—also called a **flat** data structure
 - or
 - contain records, the fields of which contain primitive types, or more records that comply with this definition recursively—called a **nearly flat** data structure
3. a recordset (table), the columns of which comply with these rules, as described in (1, flat) and (2, nearly flat)

In order to emphasize for the sake of clarity:

1	Field	Containing primitives: this is flat and complies with (1)—above
2	Record	A set of fields that comply with (1), in which case it is flat , or (2), in which case it is nearly flat
3	Recordset ()	A table, the columns of which comply with (1), in which case it is flat, or (2), nearly flat

Non-Compliant Data Structures

The following are data structures that do not comply with SAP UI parameter management:

- Records, one or more fields of which are of type recordset
- Recordsets, one or more columns of which are of type recordset

Web Service Connector Handling of Data Structures

In the event that the *Force SAP UI Parameter Management* property value is *T*, the possibilities of how variable parameters are handled by the Web Service Connector depends on the compliance of the data structure type, as defined here.

Parameter data structure is “flat”

If the parameter values returned are of **flat** data type, no special handling is required of the connector.

Parameter data structure is “nearly flat”

The Web Service Connector converts the “nearly flat” parameters to “flat.” It does this by parsing the non-primitive structures (record fields, recordset columns) into their constituent primitives, and concatenating the names, which are displayed as “flat.”

For example:

A table, Employees, has a record for each employee that comprises the fields: Employee Name (string), Employee ID (int), and Address (record).

Address, one of the columns returned by the Employee recordset, is a record comprising three fields: City (string), Street (string), HouseID. (int).

The Web Service Connector exposes the metadata from this Web service in a flat list as follows:

- Name (string)
- ID (int)
- Address_City

Connecting to Backend Systems

- Address_Street (string)
- Address_HouseID (int)

Parameter data structure contains is not “flat” or “nearly flat”

In the event that:

- a record contains a field, the data structure of which is of type recordset
or
- a recordset contains a column, the data structure of which is of type recordset

... the Web Service Connector extracts the nested (child) recordset from the record/recordset, and presents it as a new, separate input/output parameter value, and reduces 1 from the number of the record fields/recordset columns.

For example:

Taking another look at the above example of the Employees table, if the Address field were of type recordset, instead of record, the result returned would be:

- The Employees table with two columns:
 - Name (string)
 - ID (int)and
- a new parameter of type recordset, with the name 'Employees_Address', which is an Address table with three columns:
 - City
 - Street (string)
 - HouseID (int)

It should be noted that the linkage between these two recordsets is lost. In some cases, this might be useful, where the output of one of the recordsets has no business value. If, however, it is important to keep this linkage, then the *Force SAP UI Parameter Management* flag in the portal Web service system properties should be turned off (*N* is selected), Java code must be written to display the data.

When consuming such a Web service, in most cases, an automatic UI tool, such as the portal iView wizard, would support the display of only one of the output tables. The iView wizard allows you to choose which to display. Other tools may allow the display of more than one output parameter.

Reminder: In the event that the *Force SAP UI Parameter Management* property value is *N*, there are no non-compliant data types. The Web Service Connector handles all connector framework data types, except arrays.

3.8.2.4 Portal Destination Service

The Portal Destination service can be called by portal applications for obtaining connections to any available back end which can be consumed by means of connectors deployed on the NetWeaver portalConnector Framework. It is not limited only to systems in the portal system landscape (for which the Connector Gateway service is used).

Connecting to Backend Systems

The back-end connection definitions to may be stored in various system landscape repositories. As a result it brings all available connectivity landscapes within the range of the portal Connector Framework.

The Portal Destination service contains out-of-the-box implementations for connecting to: the J2EE Destination Service, WebDynpro, JCo Destinations.

- Portal System Landscape
- J2EE Destination Service
- J2EE Web Service Security Service (Dynamic Proxy)
- WebDynPro JCo Destinations

Each one of these back-end definition sources can connect to more than one type of back end. For example, the Portal System Landscape can hold connection definitions to a SAP system, a relational database system, and a Web Service. The Portal Destination service enables the client to get a list of destination names (filtered by their source and type). The client can also retrieve information for a given destination name, such as its source and type. The service also enables the client to get a Connector Framework connection to a given destination name.

The following table shows the relationship between back-end system definition sources and the Connector Framework connectors that can access them by means of the Portal Destination service.

Connector – Source Chart

Connector Type	SAP	Web Service	JDBC
Source			
Portal System Landscape	✓	✓	✓
J2EE Destination	✓	✓	
Dynamic Proxy		✓	
WebDynPro JCo	✓		



In addition to the mapping in the table, an additional connector that may be deployed to the J2EE would be mapped only to the application for which it was created.

Extending the Service

The destination service is extensible and it is possible to write and register your own implementation of system landscape repositories. See the `com.sap.portal.connectivity.destinations` package in the Javadocs for more details.

Essential Information

The Portal Destination service API is exposed as a J2EE library and therefore can be consumed from any J2EE component type. To retrieve this service instance from a portal application, add `com.sap.portal.ivs.connectorservice` in the `portalapp.xml` references list and use:

Connecting to Backend Systems

```
IDestinationsService destinationsService = (IDestinationsService)
PortalRuntime.getRuntimeResources().getService(IDestinationsService.K
EY);
```

To retrieve this service from another type of component, add to your component a reference to J2EE library **com.sap.portal.services.api** and use:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sapportals.portal.prt.registry.PortalRegistryFactory");
InitialContext context = new InitialContext(env);
IDestinationsService destinationsService = (IDestinationsService)
context.lookup(IDestinationsService.SERVICE_JNDI_NAME);
```

Code sample for using this service:

```
String[] destinationNames;
String destinationName;
IDestinationWrapper destinationWrapper;
IConnection connection = null;
DestinationFilter destinationFilter1 = new
DestinationFilter(IDestinationWrapper.SOURCE_PORTAL_SYSTEM_LANDSCAPE_
SERVICE, IDestinationWrapper.TYPE_ALL);
DestinationFilter destinationFilter2 = new
DestinationFilter(IDestinationWrapper.SOURCE_J2EE_DESTINATION_SERVICE
, IDestinationWrapper.TYPE_SAP);
DestinationFilter[] destinationFilters = new DestinationFilter[]
{destinationFilter1, destinationFilter2};
destinationNames = destinationsService.getDestinationNames(myUser,
destinationFilters);

boolean hintAreCredentialsSet;
for (int i = 0; i < destinationNames.length; i++) {
destinationName = destinationNames[i];
try {
destinationWrapper =
destinationsService.getDestinationWrapper(myUser, destinationName);
hintAreCredentialsSet =
destinationWrapper.hintAreCredentialsSet(myUser);
if (hintAreCredentialsSet) {
connection = destinationsService.getConnection(myUser,
destinationName);
} else {
// get credential (e.g. popup a logon window)
connection = destinationsService.getConnection(myUser,
destinationName, "myUserName", "myPassword");
```


Connecting to Backend Systems

```

}
// consume the back-end
} catch (Exception e) {
// error handling
} finally {
if (connection != null) {
try {
connection.close();
} catch (ResourceException e1) {
// error handling

```

3.8.2.5 Developing a Connector

The developer of a connector to a EIS is likely to be familiar with the functionality and methods of connecting and interacting with that system. The EIS-specific implementation, therefore, is up to the developer. Additional requirements are:

- The connector developer must be familiar with the **JCA standard**.
- The developer must gain familiarity with, and implement, the packages and interfaces of the portal **connector framework**, which extends the JCA standard.

For more information and details, see the sections under [Connector Framework Structure \[Page 33\]](#) and see the Javadocs for the portal and portal runtime APIs, located on the SAP Developer Network (SDN) at www.sdn.sap.com/irj/sdn/javadocs.

- The connector needs to be **deployed** on the WebAS.

See [Connector Deployment \[Page 33\]](#).

For working with the Enterprise Portal:

- The connector developer must create and deploy, in the portal, a **system template** for the EIS, upon which portal system objects, which represent the back-end system, will be based.

See [System Template \[External\]](#).



For developing connector code, though optional, it is recommended to take advantage of the new connector wizard. For details, see [Connector Wizard \[Page 33\]](#).

Prerequisites

Following are the file and environment prerequisites for the development of connectors compliant with JCA and the Enterprise Portal connector framework.

File/Environment	Description	Source
GenericConnector.jar	Required to test the connector.	Comes with SAP

Connecting to Backend Systems

		<i>Enterprise Portal.</i>
ExtendedConnector.jar	Adds functionality to generic connector	Comes with J2EE application server.
ConnectorHelper.jar	Contains utilities that support the developer.	Comes with <i>SAP Enterprise Portal</i> .
Connector.jar	Contains the JCA API.	Comes with J2EE application server.
JAAS.jar (Java Authentication and Authorization Service)	Required by the connector framework.	Comes with J2EE application server.
JTA. JAR (Java Transaction API)	Required by the connector framework.	Comes with J2EE application server.
EIS (Enterprise Information System)	Required to test the connector.	Has to be installed separately.
Java Application Server	Any J2EE JCA 1.5-compliant WAS to work with servlets without the <i>SAP Enterprise Portal</i> . Recommendation: <i>SAP Web AS 6.40 Java</i>	Has to be installed separately.
PRT (Portal Runtime)	Required to test the connector in the <i>SAP Enterprise Portal</i> environment	Comes with <i>SAP Enterprise Portal</i> .
IDE (Integrated Development Environment)	<i>SAP NetWeaver Developer Studio</i> See also Connector Wizard [Page 33] .	Has to be installed separately.
Connector Wizard	The connector wizard is a plug-in for <i>SAP NetWeaver Developer Studio</i> / Eclipse IDE. It creates a project that contains a template for a connector.	Comes with the PDK. Location: Java Developer tab -> Development -> Downloads
Connector Web Service	Required to test the connector in the <i>SAP Enterprise Portal</i> environment	Comes with <i>SAP Enterprise Portal</i> .

Additional Documentation

See also, in the Development Manual, [Connectivity and Interoperability \[External\]](#), [J2EE Connector Architecture \[External\]](#), [Implementing A 1.0 Resource Adapter \[External\]](#).

Another potentially useful source of helpful information is the *SAP J2EE Development Manual*. See *SAP Web AS for Java Applications* on the help portal.

3.8.2.5.1 Connector Wizard

The connector wizard is a plug-in for the *SAP NetWeaver Developer Studio* IDE. The connector wizard generates a new project for a connection to an EIS in the IDE that is adjusted to the capabilities of the EIS.

For more information about the plug-in installation and use, see the [Installation \[Page 33\]](#) section and [Using the Connector Wizard \[Page 33\]](#).

3.8.2.5.1.1 Installation

Obtain the file `EclipseConnectorWizard.zip` and extract it to the `plugins` folder, located under the *SAP NetWeaver Developer Studio* or Eclipse home directory and then restart the IDE. The connector wizard creates a new entry, called *SAP Connector Framework Wizard*, in the new project wizard of the IDE.

For the latest information about portal connectivity and the availability and location files for download, see SAP Note 913483.

3.8.2.5.1.2 Using the Connector Wizard

Procedure

Start the *SAP NetWeaver Developer Studio* or Eclipse. The connector wizard is started with the following steps:

4. From the IDE file command menu choose *New* → *Project*. The new project wizard dialog window is displayed.
5. Select *SAP Connector Framework Wizard* from the left pane of the new project wizard dialog window.
6. Select *Connector Wizard* from the right pane and choose *Next*.

The following dialog window is displayed:

Connecting to Backend Systems

The connector wizard expects the following user input before it can create the project:

Field Name	Description
Connector Name	Name of the connector/project. This name also serves as the base for the names of some of the classes created for the project so the name must be in accordance with the Java class naming conventions.
Connector Package	Base name for the connector packages. See the list of packages [Page 33] created by the connector wizard. The specified name must be in accordance with the Java package naming conventions.
Connector Root Folder	Root folder in which the project is stored.

7. Select the appropriate checkboxes according to the capabilities of the target EIS.

Option	Description	Created Packages (see also list of packages [Page 33])
Function Metadata Retrieval	The connector will retrieve metadata functions from the	metadata.functions metadata.primitives metadata.structures

Connecting to Backend Systems

	target application,	The class <code><c_name>InteractionSpecProperty</code> in package <code>execution.functions</code> will be generated also if the function execution checkbox is not selected.
Function Execution	The connector will execute functions of the target application	<code>execution.functions</code> <code>execution.structures</code>
Native Query Launching	The connector will implement the execution of native queries in the target application.	<code><connector_name>NativeQuery</code> in package <code>execution.objects</code> is created.
API Querying and Object Model Metadata Retrieval	The connector will retrieve the object model of the target application and implement the execution of queries.	<code><connector_name>Execution</code> in package <code>execution.objects</code> is created. Packages created: <code>metadata.objects</code> <code>execution.structures</code> <code>metadata.primitives</code> <code>metadata.structures</code>
Object Model Relations	The connector will implement the resolution of relationships of target application objects.	<code>metadata.relations</code>
Native Interface to Target Applications	The connector will retrieve a native handle to the target application API.	<code><connector_name>Query</code> in the package <code>execution.objects</code> and function <code>newQuery()</code> is created in class <code><connector_name>Connection</code> .

8. Choose *Finish* to confirm the input and to start the project generation.

A new project with the specified name and packages is created on the workbench of the IDE. The project is ready for use and can be adjusted or extended.

3.8.2.5.1.2.1 Packages Created by the Connector Wizard

Following packages and classes are created by the connector project wizard:



The names listed in `<>` refer to the names specified in the connector wizard dialog window.

`<c_package>` refers to the Connector Package.

`<c_name>` refers to Connector Name.

Connecting to Backend Systems

Package	Class
<c_package>.metadata	Capabilities.java <c_name>MetaData.java
<c_package>.connection	<c_name>Connection.java <c_name>ConnectionFactory.java <c_name>ConnectionManager.java <c_name>ConnectionMetaData.java <c_name>ManagedConnectionMetaData.java <c_name>Native.java <c_name>ManagedConnection.java <c_name>ManagedConnectionFactory.java <c_name>ConnectionSpec.java <c_name>ConnectionProperty.java <c_name>ConnectionSpecMetaData.java <c_name>DefaultConnectionPropertyGroup.java
<c_package>.execution.functions	<c_name>IndexedRecord.java <c_name>Interaction.java <c_name>InteractionSpec.java <c_name>MappedRecord.java <c_name>RecordFactory.java <c_name>InteractionSpecMetaData.java <c_name>InteractionSpecProperty.java
<c_package>.execution.objects	Query.java NativeQuery.java <c_name>Execution.java
<c_package>.execution.structures	RecordSetWrapper.java RecordWrapper.java ResultSetMetaDataWrapper.java
<c_package>.metadata.functions	Function.java FunctionException.java FunctionsMetaData.java FunctionParameter.java
<c_package>.metadata.objects	Attribute.java BusinessObject.java BusinessObjectGroup.java Method.java ObjectsMetaData.java ObjectParameter.java Index.java

Connecting to Backend Systems

<code><c_package>.metadata.primitives</code>	BigDecimalType.java BooleanType.java ByteType.java CharType.java DateType.java DoubleType.java FloatType.java IntType.java LongType.java NumberType.java ShortType.java StringType.java
<code><c_package>.metadata.structures</code>	ArrayStructure.java FieldStructure.java RecordSetStructure.java RecordStructure.java
<code><c_package>.metadata.relationships</code>	BlackBoxRelation.java DefaultRelation.java Relation.java RelationsMetaData.java



T connector wizard supplies a default implementation which may not necessarily suit the requirements of your EIS in every detail. It is important that you examine the logic of the generated classes to see if they meet the requirements.

3.8.2.5.2 Connector Deployment

Connectors, also known as resource adapters, are deployed as Java resource archives, or RAR files. The connector deployment depends on the Web application server.

SAP Enterprise Portal installation provides the following connectors:

- a [JDBC connector \[Page 33\]](#) for universal database connectivity
- an [SAP System connector \[Page 33\]](#) for connectivity to back-end SAP systems
- a [Web Service connector \[Page 33\]](#) for interacting with any Web service

These connectors are ready to use upon installation of the portal and do not require separate deployment.

Connector Archive

The Resource Adapter Archive (.RAR) file contains the following objects:

- `.jar` files that contain the connector classes – Java standard.
- `ra.xml` deployment descriptor file—which is Java standard and contains the connector metadata. See section [Deployment Descriptor Example \[Page 33\]](#).
- `manifest.mf` file that contains general version information—Java standard for RAR files. (This file is generated automatically.)
- `connector-j2ee-engine.xml` file that contains deployment information, such as component type and version, so that the application server (J2EE Engine) knows

Connecting to Backend Systems

where to deploy—for example, on portal or the J2EE server itself. This file is specific to SAP and not the Java standard for RAR's.

For information about deploying a standalone RAR file, see [Stand-Alone Deployment as RAR \[External\]](#).

See important additional information in the Reference Manual under [J2EE Engine Reference \[External\]](#) → [Deployment Descriptors \[External\]](#) → [connector-j2ee-engine.dtd \[External\]](#).

Additional Documentation

For a useful source of information on RAR deployment see, in the Development Manual, [Connectivity and Interoperability \[External\]](#) → [J2EE Connector Architecture \[External\]](#) → [Deploying the Resource Adapter \[External\]](#).

3.8.2.5.2.1 Non-Managed Mode

In the examples for the *SAP Web AS Java*, which supports JCA, the connection factory is obtained with a command such as the following:

```
ConnectionFactory conFactory =  
(ConnectionFactory) context.lookup("(ConnectionFactory)  
initctx.lookup("deployedAdapters/JDBCFactory/shareable/JDBCFactory");
```

The non-managed mode has no JNDI context to obtain a connection factory so the connector has to create a manager connection factory instance to generate a client connection factory (CCI) with the following commands:

```
JDBCManagerConnectionFactory mcf = new  
JDBCManagedConnectionFactory();  
  
ConnectionFactory conFactory =  
    (ConnectionFactory) mcf.getConnectionFactory();
```

Deployment

The `GenericConnector.JAR` file for the connector and all other JAR files required by the connector must be copied into the *global library* folder of the application server.

3.8.2.5.2.2 Deployment Descriptor Example

The name for the connector deployment descriptor file is `ra.xml`. The `ra.xml` file contains the implementation information and the connector metadata.

External libraries, that contain methods the connector references, must be placed in the *classpath* of the application server. The SAP backend system connector, for example, needs the *JCo* libraries.

`ra.xml` file example:

```
<?xml version="1.0" encoding="UTF-8"?>
```


Connecting to Backend Systems

```

<!DOCTYPE connector PUBLIC "-//Sun Microsystems, Inc.//
    DTD Connector 1.0//EN" 'http://java.sun.com/dtd/connector_1_0.dtd'>
<connector>
  <!--Basic metadata of the connector-->
    <display-name>SAP Connector</display-name>
    <vendor-name>SAP</vendor-name>
    <spec-version>1.0</spec-version>
    <eis-type>SAP</eis-type>
    <version>1.0</version>
    <resourceadapter>
      <!--Implementation of the managed connection factory interface-->
        <managedconnectionfactoryclass>
          com.sap.SAPConnector.connection.SAPConnectionFactory
        </managedconnectionfactory-class>
      <!--Interface of the connector connection factory -->
        <connectionfactoryinterface>
          javax.resource.cci.ConnectionFactory
        </connectionfactoryinterface>
      <!--Implementation of the connector connection factory interface-->
        <connectionfactory-impl-class>
          com.sap.SAPConnector.connection.CCIConnectionFactory
        </connectionfactory-impl-class>
      <!--Interface of the connector connection -->
        <connection-interface>javax.resource.cci.Connection</connection-
interface>
      <!--Implementation of the connector connection interface-->
        <connection-impl-class>
          com.sap.SAPConnector.connection.CCIConnection
        </connection-impl-class>
      <!--Connection support offered by the connector-->
        <transaction-support>NoTransaction</transaction-support>
        <authentication-mechanism>
          <authentication-mechanism-type>
            BasicPassword
          </authentication-mechanism-type>
          <credential-interface>
            javax.resource.security.PasswordCredential
          </credential-interface>
        </authentication-mechanism>
        <reauthentication-support>
          false
        </reauthentication-support>
      </resourceadapter>
    </connector>

```

3.8.2.6 Connector Web Service

The connector Web service is part of the *SAP Enterprise Portal*. The Connector Web service provides synchronous procedures, which represent the business functions, to access external clients. The Connector Web Service uses the Java API for XML-based Remote Procedure Calls (JAX-RPC) that communicates with the client over the Simple Object Access Protocol (SOAP).

Connecting to Backend Systems

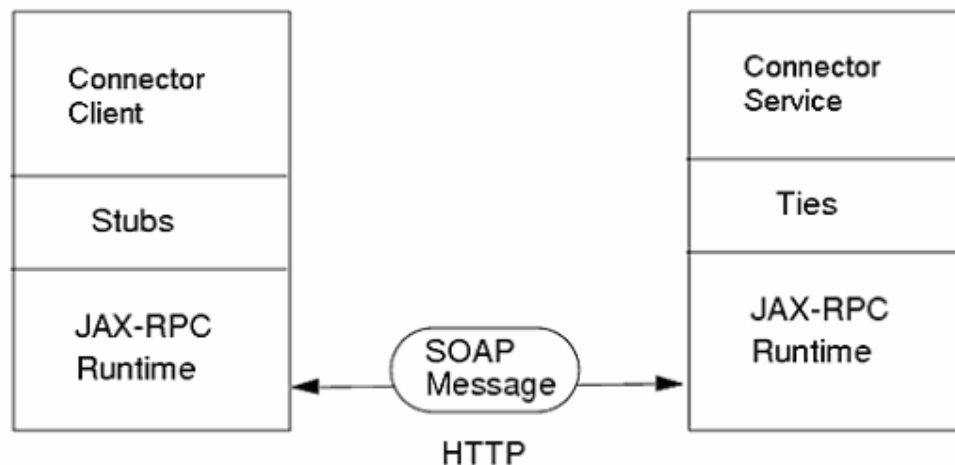
You have to be familiar with these technologies to develop a client application that uses the Connector Web service. This document includes some coding examples.

You can test the Connector Web service functions with the Test Connector Web Service Utility provided by the PDK.

Connector Web Service Architecture

To call a remote procedure, the client application invokes a method on a stub, which is a local object that represents the remote service. The stub invokes routines in the JAX-RPC runtime system of the referenced implementation. The runtime system converts the remote method call into a SOAP message and transmits the message as an HTTP request.

When the server receives the HTTP request, the JAX-RPC runtime system extracts the SOAP message from the request and translates it into a method call. The JAX-RPC runtime system invokes the method on the tied object. The tied object invokes the method on the implementation of the Connector service.



The connector service is provided by the connector framework. The Stubs and ties are generated by the *xrpsc* tool that comes with JAX-RPC. JAX-RPC comes with the WAS.

Using the Connector Web Service

The connector client is a stand-alone program that calls the `ExecuteCommand` method of the `ExecuteCommand` service. It makes this call over a stub, a local object which acts as a proxy for the remote service. The client application can be written in any language that is supported by SOAP messaging.

Client Application Workflow

9. Create the URL endpoint.
10. Create the SOAP message with header and set the method name and input parameters.
11. Send the message and a receiving reply.
12. Parse and extract the output.

Connecting to Backend Systems

See section [Client Application Coding Example \[Page 33\]](#).

Deployment of Web Services

The Connector Web Service is a PRT service and installed with the *SAP Enterprise Portal*.

3.8.2.6.1 Client Application Coding Example

```
// Establish the URL endpoint which is the connector Web service.

to = reqBase + "/irj/servlet/prt/soap/ConnectorWebService";
URLConnection urlEndpoint = new URLConnection(to);
int id = Integer.parseInt(id_msg);

// Construct the SOAP message.
SOAPMessage msg = constructMessage(id);

// Send the message to the provider using the connection.
SOAPMessage reply = con.call(msg, urlEndpoint);

// Constructing the headers of the SOAP message.
// (according to your requirements).
// A common header declaration, as shown here, is defining namespaces
.
// Create a message factory.
MessageFactory mf = MessageFactory.newInstance();

// Create a message from the message factory.
SOAPMessage msg = mf.createMessage();

// Message creation takes care of creating the SOAPPart,
// a required part of the message as per the SOAP 1.1 specification.
SOAPPart sp = msg.getSOAPPart();

// Retrieve the envelope from the soap part to start building the
// soap message.
SOAPEnvelope envelope = sp.getEnvelope();

// Create a soap header from the envelope.
SOAPHeader hdr = envelope.getHeader();

// get a soap body from the envelope.
SOAPBody bdy = envelope.getBody();

bdy.setEncodingStyle("http://schemas.xmlsoap.org/soap/encoding/");
bdy.addNamespaceDeclaration("wn0", wn0URI);
bdy.addNamespaceDeclaration("wn1", "http://www.w3.org/2001/XMLSchema");
bdy.addNamespaceDeclaration("wn2",
                             "http://www.w3.org/2000/10/XMLSchema");
;
bdy.addNamespaceDeclaration("wn3", "http://www.w3.org/1999/XMLSchema");
```

Connecting to Backend Systems

```

    bdy.addNamespaceDeclaration("prt", "http://prt.java.soap/schemas")
;
    bdy.addNamespaceDeclaration("wn4",
                                "urn:com.sap.ConnectorWebService");
    bdy.addNamespaceDeclaration("xsi", xsiURI);

// You generate the rest of the SOAP message according to the method
// you want the Web server to run. (Note: Only one method per request
//.)
// For example: getting connector capabilities.
    Name childName = null;

// Create a Name object for the method.
    Name methodName = envelope.createName("getCapabilities", "wn4", null);

// Add a soap body element to the soap body.
    SOAPBodyElement service = bdy.addBodyElement(methodName);

    childName = envelope.createName("JNDIName");
    SOAPElement param0 = service.addChildElement(childName);
    Name typeName = envelope.createName("type", "xsi", xsiURI);
    param0.addAttribute(typeName, "wn1:string");
    param0.addTextNode("EISConnections/JDBCFactory");

    childName = envelope.createName("connectionString");

    SOAPElement param1 = service.addChildElement(childName);
    typeName = envelope.createName("type", "xsi", xsiURI);
    param1.addAttribute(typeName, "wn1:string");
    param1.addTextNode("driver=com.sap.jdbc.sqlserver.SQLServerDriver,
                        url=jdbc:sap:sqlserver://localhost:1433;
                        user=sa;password=sa;databaseName=Northwind,");

// Example for launching a query:
private void executeQuery(SOAPEnvelope envelope, SOAPBody bdy)
    throws Exception
{
    Name childName = null;
    Name methodName = envelope.createName("executeQuery", "wn4", null);

// Add a soap body element to the soap body.
    SOAPBodyElement service = bdy.addBodyElement(methodName);

    childName = envelope.createName("JNDIName");
    SOAPElement param0 = service.addChildElement(childName);
    Name typeName = envelope.createName("type", "xsi", xsiURI);
    param0.addAttribute(typeName, "wn1:string");
    param0.addTextNode("EISConnections/JDBCFactory");

    childName = envelope.createName("connectionString");
    SOAPElement param1 = service.addChildElement(childName);
    typeName = envelope.createName("type", "xsi", xsiURI);
    param1.addAttribute(typeName, "wn1:string");

    param1.addTextNode("driver=com.sap.jdbc.sqlserver.SQLServerDriver
,

```

Connecting to Backend Systems

```
url=jdbc:sap:sqlserver://localhost:1433;  
user=sa;password=sa;databaseName=Northwind,");  
  
childName = envelope.createName("queryString");  
SOAPElement param2 = service.addChildElement(childName);  
typeName = envelope.createName("type", "xsi", xsiURI);  
param2.addAttribute(typeName, "wn1:string");  
param2.addTextNode("select * from [dbo].[Customers]");  
  
childName = envelope.createName("startFromRecord");  
SOAPElement param3 = service.addChildElement(childName);  
typeName = envelope.createName("type", "xsi", xsiURI);  
param3.addAttribute(typeName, "wn1:int");  
param3.addTextNode("1");  
  
childName = envelope.createName("recordsNumberToRetrive");  
SOAPElement param4 = service.addChildElement(childName);  
typeName = envelope.createName("type", "xsi", xsiURI);  
param4.addAttribute(typeName, "wn1:int");  
param4.addTextNode("10");  
}
```

3.8.2.6.2 Test Connector Web Service Utility

The *Test Connector Web Service Utility* is a PDK tool to test the connector Web service functions by running connectivity tests to existing systems. The tool can launch a function call or an SQL query, whatever the selected system supports. The tool displays the SOAP request and the returned response in separate areas. The returned response is either the expected output or an appropriate message.

Using the Test Connector Web Service Utility

Testing a Function

1. From the PDK top level navigation choose **Java Developer** and then the following commands:

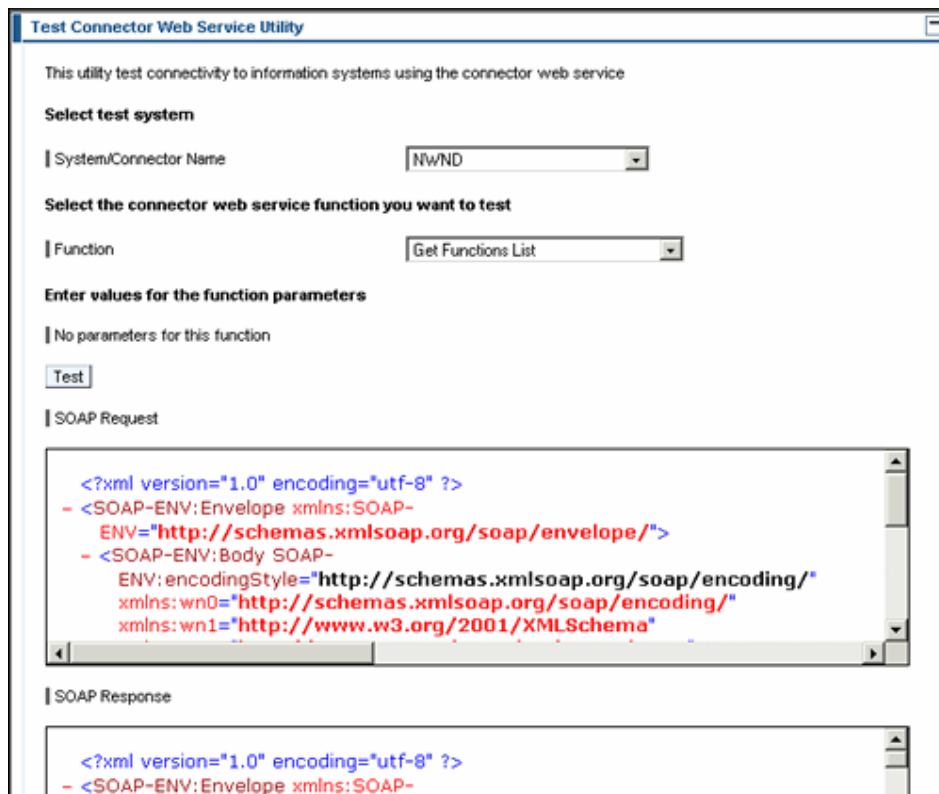
Development → Support Desk

The support desk iView is opened in the content area.

2. From the *Connection Testing* area in the support desk choose *Connector Web Service*.

The *Test Connector Web Service Utility* is displayed in a separate browser window and has the following user interface:

Connecting to Backend Systems



- Under *Select test system* choose an existing system or installed connector in the *System/Connector Name* dropdown list box.

If you choose a connector, a *Connection String* input field is displayed that contains the connection parameters. The correct parameters have to be provided manually.

Exampe of a connection string for *SAPFactory_LoadBalancing*:

```
Group=, User=, client=, Language=, gwserv=, mshost=, tphost= ...
```

Example of a connection string for the JDBC connector:

```
driver=com.sap.portals.jdbc.sqlserver.SQLServerDriver.
url= jdbc:sap:sqlserver:

//<server>:1433;databaseName=Northwind,User=sa,Password=adm
in,
```

- From the *Function* dropdown list box choose the Web service function you want to test. The Web Service function you choose must be supported by the system chosen before.

Availabel Web service functions

Function	Discription
Get Capabilites	Gets the meta data of the system or connector.

Connecting to Backend Systems

Get Root Business Object Group	Gets the name of the business tree object root.
Get All Business Objects	Gets all bottom level nodes of the business object tree, for example, from a database table.
Get Business Object Tree	Gets all groups and objects from the business object tree.
Get Business Object	Gets the meta data of the business object, for example a field of a database.
Get Root Business Object Group Children	Gets the content of a specific business object group.
Execute Query	Launches an SQL query.
Get Function List	Database: Get all stored procedures. Backend system: Get all RFC
Get Function	Gets the parameters for the specified function.
Execute Function	Tests the execution of the specified function.

5. If the chosen function does not parameters, the message *No parameters for this function* is displayed.

If the function chosen requires parameters, the parameters have to be entered with the appropriate values in the following format:

```
<par1_name>=<par1_value>,<par2_name>=<par2_value>,...
```

Testing a Query

1. Choose *Execute Query* from the *Function* dropdown list box and enter a valid SQL query in the *Query String* text area.
2. Enter the row number where the returned *RecordSet* should start.
3. Enter the number of rows to be returned.

Query String	<input type="text" value="Select * from orders;"/>
Record to start	<input type="text" value="10"/>
Number Of Records	<input type="text" value="5"/>

Start the Test

Choose the *Test* button and wait for a response. The request is displayed in the *SOAP Request* area and the test response or message is displayed in the *SOAP Response* area.

3.8.3 Dynamic System Resolution

Purpose

The Dynamic System Resolution feature enables you to create a service that selects at runtime the system to which an iView connects.

Generally, an iView connects to a system that is defined at design time. With dynamic system resolution, an iView can select at runtime from among several systems, based on the location of the current user, network traffic or any other logic.

For example, for a specific iView, you may want users in Europe to connect to a system in Germany while users in the United States connect to a similar system in New York.

How System Resolution Works

An iView that needs to connect to a back-end system contains a reference to a system alias. The iView — via the Connector Gateway or other service — calls the system landscape service method `getSystemId()` in order to resolve the alias to a specific system defined in the Portal Content Directory (PCD). By default, the system landscape service simply queries the PCD for the system that is associated with that alias, and returns the PCD path to that system.

In the PCD, a system alias is associated with only one system.

In order to resolve an alias to a different system, you can create a system resolving service for a specific alias. When an iView calls `getSystemId()` to resolve the alias, the system landscape service first checks if a custom service exists for the alias. If one exists, the system landscape service calls the resolving service — instead of querying the PCD — in order to get the corresponding system.

The resolving service — which receives from the system landscape service the name of the alias and a reference to the current user — can resolve the alias to any system based on any logic.

This section contains the following information:

- [Dynamic System Resolution Workflow \[Page 33\]](#): Describes what happens when an iView requests a system via an alias that must be resolved dynamically.
- [Writing a System Resolving Service \[Page 33\]](#): Describes how to write a system resolving service.
- [Checking Deployment \[Page 33\]](#): Describes how to check the deployment of a system resolving service.
- [Removing a System Resolving Service \[Page 33\]](#): Describes how to remove a system resolving service.

3.8.3.1 Dynamic System Resolution Workflow

The following describes the process for resolving any system alias:

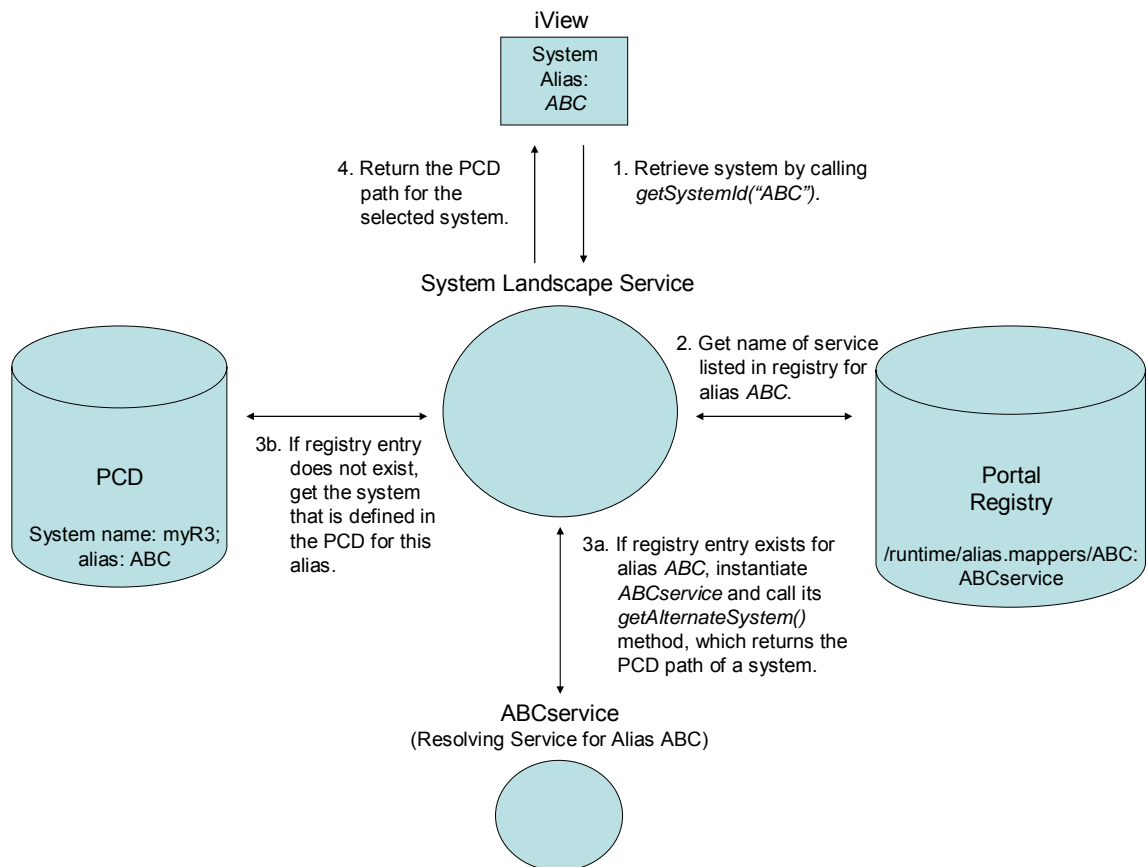
1. A portal component calls the `getSystemId()` method in the system landscape service in order to resolve an alias to the PCD path for a system.

Connecting to Backend Systems

2. The system landscape service checks the portal registry under `/ROOT/runtime/alias.mappers` for an entry with the same name as the alias. This entry contains the name of the service for resolving the alias.
3. If there is an entry, the system landscape service instantiates the service listed in the registry for the specified alias, and calls the `getAlternateSystem()` method, providing to the method the current user as an `IUser` object and the name of the alias as a string. The method returns a string that represents the PCD path of a system.
If there is no entry, the system landscape queries the PCD for the system to which the alias is associated.
4. The system landscape service returns the PCD path as a string to the portal component.

The following figure illustrates the process for a case when:

- An iView tries to connect to a system with alias `ABC`.
- A resolving service named `ABCservice` is created and uploaded to the portal.
- A registry entry is created called `ABC` with a value of `ABCservice`, which is the name of the service for resolving the `ABC` alias.



3.8.3.2 Writing a System Resolving Service

A portal component that contains a system resolving service is composed of two parts:

- A java class that implements the following interfaces:
 - **IService**: The standard interface for all services
 - **IDynamicSystemService**: The interface for a system resolving service. This interface exposes one method, `getAlternateSystem()`, which returns a string that represents the PCD path to a system defined in the PCD.
- A *portalapp.xml* file that defines the following:
 - The entries in the portal registry to create in order to indicate to the system landscape service for which aliases to invoke this service
 - The application properties that allow the service to function as a resolving service
 - The name of the resolving service and the class within the PAR that implements the service

A portal component can contain more than one system resolving service, if the component PAR file contains several java classes that implement the `IDynamicSystemService` interface.

A system resolving service can resolve more than one alias, if the *portalapp.xml* file creates several registry entries for the service.

3.8.3.2.1 IDynamicSystemService

Every system resolving service must implement `IDynamicSystemService`, which exposes a single method, `getAlternativeSystem()`.

IDynamicSystemService

Method	Description	Arguments	Return value
<code>getAlternativeSystem</code>	Returns as a string the PCD path for the given alias	(<code>IUser user</code> , <code>String alias</code>)	<code>Java.lang.String</code>

Example

The following shows the class declaration and the `getAlternativeSystem()` method for implementing the `IDynamicSystemService` interface.

The methods for implementing the `IService` interface are not shown.

```
public class MyAliasMapping implements IDynamicSystemService, IService {
    .
    .
    .
    public String getAlternativeSystem(IUser user, String alias)
    {
        String system = "pcd:portal_content/systems/db2_us";
        return system;
    }
}
```

The `getAlternativeSystem()` method provides the logic for resolving an alias to the PCD path of a system. Generally, this logic is based on the user and the system alias that are passed to the method.

Connecting to Backend Systems

The user object that is passed into the `getAlternativeSystem()` method is of type `IUser`. You can query the `IUser` object for such things as the user's groups and roles. For more information on the `IUser` interface, see the User Management Engine documentation in the Portal Development Kit.

3.8.3.2.2 portalapp.xml

The `portalapp.xml` file for a system resolving service contains the following sections:

- **Registry:** Creates entries in the portal registry to indicate to the system landscape service the aliases for which to invoke this service.
- **Application Configuration:** Specifies the properties that allow the service to function as a resolving service.
- **Services:** Specifies the name of the resolving service and the class within the PAR that implements the service.

The following is an example of a `portalapp.xml` file for a system resolving service:

```
<?xml version="1.0" encoding="utf-8"?>
<application>
  <registry>
    <entry path="/runtime/alias.mappers/pubs" name="pubs_srv"
type="service"/>
  </registry>
  <application-config>
    <property name="releasable" value="false"/>
    <property name="startup" value="true"/>
    <property name="ServicesReference"
value="com.sap.portal.ivs.api_dynamicSystemService"/>
  </application-config>
  <components/>
  <services>
    <service name="pubs_srv">
      <service-config>
        <property name="className"
value="com.sap.portal.dynamicsystem.sample.MyAliasMapping"/>
        <property name="startup" value="true"/>
      </service-config>
    </service>
  </services>
</application>
```

Registry Element

The `<registry>` element of the `portalapp.xml` file specifies the aliases for which the services defined in this PAR will be invoked.

```
<registry>
  <entry path="/runtime/alias.mappers/pubs" name="pubs_srv"
type="service"/>
</registry>
```

The `<registry>` element contains one or more `<entry>` elements. Each `<entry>` element contains the following attributes:

- **path:** A key to create in the portal registry. The key name should be the name of a system alias to be resolved. The key should always be placed in the registry under

Connecting to Backend Systems

/ROOT/runtime/alias.mappers. In the above example, the alias pubs is specified.

- **name:** The name of the service to invoke when a system is requested for the alias referred to in the `path` attribute. The names of the services included in the PAR are defined in the `<services>` element.
- **type:** Always set to `service`.

Application Configuration Element

The `<application-config>` element sets the properties of the PAR file so that the services defined in the PAR can function as system resolving services. The section should always be written as it appears below.

```
<application-config>
  <property name="releasable" value="false"/>
  <property name="startup" value="true"/>
  <property name="ServicesReference"
value="com.sap.portal.ivs.api_dynamicSystemService"/>
</application-config>
```

The `<application-config>` element contains the following `<property>` elements:

- **releasable:** Prevents the service from being dropped by the system, for example, if the system runs low of memory.
- **startup:** Specifies that the application is initialized at startup and deployed locally, improving performance.
- **ServicesReference:** References the dynamic system service, which provides the API for this feature.

Services Element

The services element specifies the system resolving services defined in the PAR.

```
<services>
  <service name="pubs_srv">
    <service-config>
      <property name="className"
value="com.sap.portal.dynamicsystem.sample.MyAliasMapping"/>
      <property name="startup" value="true"/>
    </service-config>
  </service>
</services>
```

Each service is described by a `<service>` element, whose `name` attribute specifies the name of the service.

Each `<service>` element contains a `<service-config>` element, which contains the following `<property>` elements:

- **className:** The implementing class for the system resolving service.
- **startup:** Indicates to start the service when the portal runtime is started. Always set this property to `true`.

3.8.3.2.3 User Mapping

When connecting to a system, an iView must provide a user name and password. The iView queries the PCD for the default alias of the system to which it wants to connect, then retrieves the user name and password mapped for the current user for this alias.

Connecting to Backend Systems

With dynamic system resolution, a resolving service is allowed to select any valid system defined in the PCD, even one without an alias. However, a system without an alias cannot have a user mapping and, therefore, no iView could connect to such a system.



Even with dynamic system resolution, make sure that all the systems in the PCD that are required by your content have aliases and user mappings for their respective back-end systems.

3.8.3.3 Checking Deployment

After deploying a resolving service to the portal, you can check whether the service is properly registered and started and whether the aliases to be resolved by this service are properly registered.

Checking a Resolving Service

To check that a service is registered and started, navigate to the Portal Runtime Application Console as follows:

- *System Administrator* → *Support* → *Support Desk* → *Port Runtime* → *Application Console*

From the dropdown list, select a PAR that contains resolving services and click **Show**. The console shows the services contained in the PAR, and whether each service is started.

Checking Registered Aliases

To check that an alias is registered to be resolved by a resolving service, navigate to the Portal Registry Browser as follows:

- *System Administrator* → *Support* → *Support Desk* → *Port Runtime* → *Portal Registry Browser*

In the registry browser, navigate to **/ROOT/runtime/alias.mappers**. The browser shows the aliases that are registered for dynamic system resolution. For each alias, the fully qualified name of the service (that is, the name of the service preceded by the name of its PAR) is displayed.

3.8.3.4 Removing a System Resolving Service

To remove a system resolving service, remove from the portal the PAR that contains the service.



When you remove a PAR, you remove all resolving services defined in that PAR.

If a PAR contains a service that resolves more than one alias or a PAR defines more than one service, you may need to modify the PAR and then redeploy it instead of simply removing the PAR.

Removing a Service

To remove a service from a PAR that contains two services, do the following:

1. Remove from the PAR the class file that defines the resolving service that you want to remove.

Specialities in the Portal

2. Modify the *portalapp.xml* file for the PAR by doing the following:
 - Remove the `<service>` element for the service that you want to remove.
 - Remove the `<registry>` elements for the aliases that were to be resolved using the service that you want to remove.
3. Redeploy the PAR.

Removing an Alias for a Service

A service can be configured to resolve two or more aliases. To prevent the service from resolving one of the aliases, do the following:

1. Modify the *portalapp.xml* file for the PAR by removing the `<registry>` element for the alias that you no longer want resolved by the service.
2. Redeploy the PAR.

3.9 Specialities in the Portal

This section contains special topics about developing for the portal.

3.9.1 Implementing an External-Facing Portal

Purpose

The portal includes a set of features that enable the use of the portal as an external-facing portal, that is, a public Web site. These features provide for the following:

- **Improved performance over the internet** by reducing the number of resources required for each request. An external-facing portal avoids the use of resource-heavy HTMLB and client-side eventing (EPCM).
- **Web-like behavior**, including the use of standard browser buttons, such as *Back*, *Forward* and *Refresh*, for navigation.
- **Easy customization of the portal look and feel** with the use of tag libraries for creating navigation iViews and page layouts with custom iView trays.
- For more information on implementing an external-facing portal, see [Implementing an External-Facing Portal \[External\]](#) in the *Portal Administration Guide*.

Developer Tasks

The following portal features are relevant to developers when implementing an external-facing portal:

- **Light Framework Page:** The portal provides a light framework page with the following characteristics:
 - Includes light navigation iViews that do not use HTMLB or client-side eventing (EPCM) JavaScript, reducing the resources required for rendering the page.
 - Renders the portal in a single frame, which enables Web-like behavior, such as the use of the browser navigation buttons. The single frame also eliminates the need for JavaScript to enable communication between frames.

- For more information on building your own light navigation iViews, see [Creating Navigation iViews \[Page 33\]](#).

- **Resource-Sensitive Page Builder:** The page builder can prevent the downloading of client-side eventing JavaScript if an iView indicates it does not need eventing. The need for eventing is specified by a component's `EPCFLevel` profile property.

For more information on this property, see [Components \[Page 11\]](#) in the developer documentation.

- **JSP Tag Libraries:** The portal provides tag libraries for easily developing navigation iViews and page layouts from JSP pages. The following tag libraries are provided:

- **Navigation Tag Library:** Enables you to develop iViews that access the current user's navigation hierarchy. The tag library is generally used for top-level and detailed navigation iViews.

For more information, see [Navigation Tag Library \[Page 33\]](#).

- **Framework Tag Library:** Enables the creation of links for various portal functions, such as logging off or personalizing the portal. The tag library is generally used for masthead and page title bar iViews.

For more information, see [Framework Tag Library \[Page 33\]](#).

- **Layout Tag Library:** Enables you to develop page layouts, and to develop custom iView trays.

For more information, see [Layout Tag Library \[Page 33\]](#).

- **Navigation Cache and Quick Links:** You can implement navigation caching and quick links for navigation hierarchies that your navigation connectors create.

For more information, see [Creating Navigation Connectors \[Page 33\]](#).

4 Ensuring Quality

Purpose

This section provides information about how to improve the quality of your portal applications, and includes the following:

- [Developing Well Performing Portal Applications \[Page 33\]](#)
- [General Rules and Guidelines for Managing Exceptions \[Page 33\]](#)

4.1 Developing Well Performing Portal Applications

Purpose

The *SAP NetWeaver Developer Studio* provides tools for the development of portal applications and services for the *SAP Enterprise Portal*. However the complexity of the Java language still leaves a lot of room to create applications that affect the overall performance of the portal significantly.

This guide discusses performance issues of portal applications and also gives an inside view of the general nature of the Java execution environment. It can be used as an anti-pattern

Developing Well Performing Portal Applications

reference during development and as a handbook for code reviews. After bottlenecks/hotspots have been identified, the different topics can be inspected for matching anti-patterns that may help to solve the issues.

This document can be used as follows:

- Before you start developing, to get suggestions for certain tasks.
- During the development phase, to avoid performance problems later.
- After the development phase, when applications do not have the expected performance.

The hints in this guide are highlighted with the hint symbol. First important hint:



Do not create *Threads* yourself

This guide also contains additional rules for the *JLin* plug-in in the *SAP Netweaver Development Studio*. These rules will help you to spot places in your code that need improvement. The performance issues found in the various applications now in use at the different customers of the portal are similar.

The *JLin* rules are marked as follows:



Use *JLin* rule ... to spot suspicious coding.

Automatic source code inspection and to use the options properly is delicate job. Depending on the very individual circumstances not every warning is justified and it is necessary to bypass and ignore *JLin* warnings using special comments.

4.1.1 Server Side Programming

Application servers are resource sharing environments. All applications and services running on a server node use the same memory, the same CPU and often the same database system. This scenario is very similar to the multi-user environment of a mainframe.

Unfortunately, Java differs from the mainframe world in one crucial point:

Java does not enforcement of resource limits upon the applications. There are no quotas for CPU-time or allocated application memory. As a result, every single application may bring down the entire server node.



Resources are shared by all applications on a server node. There is no limit for an application how many resources it can use.

4.1.1.1 Memory Usage

Memory consumption is a constant issue in server environments. The Java language itself is one of the major reasons for memory problems because it fosters the creation of many temporary objects on the heap. There is also a certain overhead due to the general usage of Unicode and automatic memory management (garbage collection).

An application uses the memory resource as follows:

Developing Well Performing Portal Applications

- Memory footprint

The amount of memory an application requires during the entire lifecycle in order to keep its state (data).

- Temporary memory demand

The amount of memory an application requires to perform a specific operation that is released afterwards. This memory is used for temporary data.

- Garbage throughput

The garbage throughput is the temporary memory demand for a unit that is allocated and released shortly thereafter, measured in time units. This memory consumption comes from calculations, like a *StringBuffer* or a *Calendar*. There is a direct connection between the garbage throughput and the temporary memory demand.

How an application uses memory depends on several factors. For the memory footprint it is important, how much data is kept in memory and how much data is stored in a database or files.

The temporary memory demand is highly dependent on the used algorithm to perform a certain operation. Sometimes execution can be accelerated by loading all or a big portion of the data to be processed into memory, do the processing and store the results in a database or file. For an isolated view, this is the fastest and easiest approach in the majority of cases. For the server the approach may be completely different. Because of insufficient memory resources, competing applications are slowed down significantly. Other applications have to work "on the disk" because caches are flushed too early.

4.1.1.2 CPU Usage and Threads

Like the memory also the CPU is a shared resource. The good news is that the JVM assigns processing time to the different competing applications. Nevertheless processing efficiency is still very important to the performance and scalability of server applications.

Problems may arise when the scheduling system itself is overloaded by creating too many threads. The management of the threads is taking a significant portion of memory and CPU time.

Conventions to ensure system stability and responsiveness:

- Make sure that parallel processing brings a benefit in a multi-user scenario. Additional background jobs are useless, when the CPU load is already high,
- Do not create threads from portal applications.
- Dedicate parallel processing/background tasks to dedicated services, using Message-Driven Beans.

Performance increases with additional threads in a single user scenario is different to a multi user scenario. A system with high CPU load will not benefit from additional threads.



Do not create *Threads* yourself.

Thread creation is one of the core responsibilities of the J2EE container and the portal on top of it. It is strongly interconnected with the transaction and context management. Therefore the creation of threads is not allowed for applications.

An application only benefits from parallel processing when the CPU would be idle otherwise. *Message Driven Beans* is the technology that should be used to parallelize tasks.

Message-Driven Beans

The most common scenario for parallel processing is running a query on multiple external systems like databases or SAP backend systems. For every query i an I/O request is sent to the external system and the CPU is waiting time t_i for the response to arrive. Without parallel processing this comes to a total of $t_1 + \dots + t_k$. With concurrency this can be reduced to the maximum of the response times $\max(t_1, \dots, t_k)$ – a big difference, when the response times can not be ignored.

A message-driven bean is an Enterprise Java Beans (EJB) that is executed asynchronously. A message-driven bean can be used to implement concurrent processing for applications. For parallel task a message-driven bean is created that will process the tasks in incoming order with a adjustable number of concurrent working threads. The results of the tasks are put in a result queue that can be accessed by the client with the JMS API.



JLin rule Thread Creation detects the creation of new threads by portal applications.

4.1.1.3 Scalability

Once a portal is productive, deployed applications often experience a huge increase in the volume of managed data and the number of concurrent users on the system. It is important that the production environment can keep up with the increasing load by adding memory, CPUs and new cluster nodes.

The following rule of thumb is a good approximation of what scalability is all about. Scalability can be considered as the increase of the capacity (for example, number of concurrent users and transactions per second) divided by the required investments in hardware (for example, number of processors, total memory, number of cluster nodes). The goal is of course to maximize the scalability value also for big installations.

$$\text{Scalability} = \Delta\text{Capacity} / \Delta\text{Hardware}$$

Scalability indicates how much a system can benefit from added resources. When adding a CPU to a system, it is important that a maximum portion of the processing work can be done parallel. Operations that require exclusive access to a resource will create a bottleneck, because only one thread and therefore one CPU may access it and the others will spend their time waiting for it to be released. Therefore all data that is shared between users, applications, services or sessions can interfere with scalability.

Improving scalability:

- Evaluate data structures and algorithms with large amounts of data.
- Limit synchronization to the smallest scope possible.
- Avoid per-request I/O-operations whenever possible.

4.1.1.4 Caching and Deferred Execution

Also software applications follow the famous 80:20 rule. They spent 80% of the execution time in 20% of their operations. In other words: Only a small part of the code-base is relevant for the performance of a software system. Portal applications spent most of the time for I/O operations, for example, accessing a database or communicating with an external system.

Developing Well Performing Portal Applications

Therefore I/O-intensive operations have to be used with care, especially with the following APIs:

- PCD/Portal Content Directory
- UME/User Management Engine
- KM/Knowledge Management
- JCo/Java Connector for SAP backend system

These APIs involve a lot of database processing and that leads to long execution times. Therefore, they should be avoided and/or consolidated whenever possible. Use the following strategies to reduce I/O operations:

- Delay expensive operations and avoid preloading.
- Keep information in memory that is likely to be queried again.
- Consolidate multiple operations to one (for example, SQL statements).
- Avoid file I/O in requests.

4.1.2 Java Programming

This chapter covers aspects of application performance that apply to any Java application, independent from the execution environment, and suggests general programming rules for Java development.

For deeper coverage of this topic, please refer to the following books:

- *Effective Java* by Joshua Bloch [\[1\] \[Page 33\]](#)
- *Java Performance Tuning* by Jack Shirazi [\[2\] \[Page 33\]](#)

4.1.2.1 Program Flow

The Check-and-Get Idiom

A very common idiom is the check-and-get code pattern. The code pattern has the following steps:

- Check for availability of data that match a query criteria.
- Retrieve the data that match the query criteria.

While the approach to check the availability of data before getting the data is straightforward, it has a bad impact on application performance.

Example: Check-and-get in maps

```
if (map.containsKey(myKey)) { // check
    myValue = map.get(myKey); // get
    ..
}
```

The problem with this idiom is that the *get* operation always includes the activity that is performed by the *check* operation. The difference is that the latter does a lookup (Example for lookup: finding a row in a table, finding a key in a hash-table, locating an object in a list) but throws away the results. As (in the idiom) the *get* operation is always called when *check* was successful, we have a duplication of efforts in the positive case. The negative case (*check* fails) does not compensate this drawback, since a *get* operation returning no results is (in most cases) just as complex as a *check* operation returning *false*.



Avoid Extra Checks

Checks like *hasChildren*, *isEmpty*, *contains* and so on should be avoided, whenever the data/objects whose availability is checked are retrieved later anyway. This is in particular the case when the availability check involves complex processing like in database-access or access to connected systems.

Examples for redundant checks (check-and-get idiom)

Applies to	Check Operation	Get Operation
<code>java.util.Map</code>	<code>Contains</code>	<code>get</code>
<code>java.util.Collection</code>	<code>isEmpty</code>	<code>Iterator</code>
<code>com.sapportals.wcm.repository ICollection</code>	<code>hasChildren</code>	<code>getChildren</code>
SQL	<code>select COUNT(ID) from ... where</code>	<code>Select ID, COL1, COL2, COLn from ... where</code>

Null-Checking and NullPointerException

The creation and throwing of exceptions is an expensive operation in Java. Therefore exceptions should not be provoked without good reason.

Some programmers avoid explicit *null*-checks by catching the corresponding runtime exception *NullPointerException*, thinking the source code is executed faster because an *if*-statement has been saved. In fact the cost of the exception is so huge that a single null value in 1.000.000 cases will be enough to eradicate the performance advantage of the saved *if* statements. So this technique does not save time but will lead to noticeable performance degradation when *null* values occur more often.

Conclusion: *NullPointerExceptions* should serve their purpose to indicate programming errors and nothing else.



The *JLin* rule *NullPointerException Check* will find the catch blocks for *NullPointerExceptions*.

Loops

Counting Loops vs. Iterators

The common denominator of all Collection Classes is the interface *Collection*, which is implemented for all lists and sets and for the keys and values of maps. All collections support sequential iteration using the *Iterator* interface.

The traditional C-style iteration idiom using an integer index has no advantage in terms of performance or memory requirements unless the index is also used for another purpose. The following example shows a popular idiom.

```
for (int i=0; i < list.size(); i++) {  
    doSomething(list.get(i));  
}
```

The example has following flaws:

- The constant expression `size()` is repeatedly evaluated.
- The class of the `list` object is required to implement the `List` interface (strong precondition)
- This iteration pattern is very slow for *LinkedLists* and other implementations that do not offer fast random access (indicated by the *RandomAccess* interface).



Use index-based access only when needed. Index-based access often precludes the use of efficient data structures based on hash-codes or balanced trees.

Following implementation alternatives for the example:

```
for(final Iterator iter = list.iterator(); iter.hasNext();) {  
    doSomething(iter.next());  
}
```

or, if index-based access is needed:

```
for(int i = 0, n = list.size(); i < n; i++) {  
    doSomething(list.get(i));  
}
```

Iterating through maps

A often ignored method in the *Map* interface is the *entrySet* method. Many applications emulate the *entrySet* method by using the *keySet* method and *get* for every entry in the set:

```
Iterator iter = map.keySet().iterator();  
while (iter.hasNext()) {  
    Object key = iter.next();  
    Object value = map.get(key);  
    ...  
}
```

This work around does the job but the performance is less not acceptable. Consider that the lookup operation *get* is called for every element in the map.

We recommend the following implementation that works without a lookup:

```
Iterator iter = map.entrySet().iterator();
while (iter.hasNext()) {
    Map.Entry e = (Map.Entry) iter.next();
    Object value = e.getValue();
    ...
}
```



Use the JLin Rule *Map Iteration* to find suspicious places in your source code.

Repeated Evaluation of Invariant Expressions

Everything inside a loop is processed with every loop. Therefore the loop must contain only statements that have to be processed in the loop. Each expression that does not change during the loop execution should be evaluated and assigned to a local variable before the loop is entered.

```
for (Iterator iter = list.iterator(); iter.hasNext();) {
    MyObject obj = (MyObject) iter.next();
    PropertyName pn = new PropertyName(
        bean.getPortalCustomNamespace(),
        Settings.RES_PROPERTY);

    if (obj.getProperty(pn).toString()
        .equalsIgnoreCase(Settings.RES_THUMB) {
        ...
    }
}
```

The statement that violates the rule is highlighted in this example. The only variable that changes on every cycle is the *obj* variable. Since this variable is not a parameter of the creation of the *PropertyName* object we can be sure that the value of variable *pn* is the same for every iteration cycle.

We can also rule out that *pn* is changed in the cycle, so we can be sure that the meaning of the loop is not changed when we use the same *PropertyName* instance on every iteration instead of several *equal* ones.

```

    PropertyName pn = new PropertyName(
        bean.getPortalCustomNamespace(),
        Settings.RES_PROPERTY);

    for (Iterator iter = list.iterator(); iter.hasNext();) {
        MyObject obj = (MyObject) iter.next();
        if (obj.getProperty(pn).toString()
            .equalsIgnoreCase(Settings.RES_THUMB) {
            ...
        }
    }
}

```

4.1.2.2 Build-In Types (java.lang)

Strings

Concatenation

Strings are easy to use in Java because they have a built-in concatenation operation and are thread-safe due to their immutability. This advantage can also become a major bottleneck.

The string concatenation operator `+` is a convenient way to combine a few strings into one. It is easy to use for generating a single line of output or for constructing the string representation of a small, fixed-size object. Unfortunately the immutability has some significant side effects:

- For every concatenation a new string instance has to be created (creating garbage objects).
- All characters, not only those that are added, have to be copied to the new string. This leads to *quadratic complexity* when concatenating in a loop.

Example: Consider the following method that constructs a string representation of a statement by concatenating a line for every item:

```

public String statement() {
    String s = "";
    int n = numItems();
    for (int i = 0; i < lines; i++) {
        s += lineForItem(i);
    }
    return s;
}

```

This kind of implementation will have to perform n^2 copy-character operations during the creation of a result string s of length n . Therefore the method performs very badly with a large number of items. By concatenating the strings using a *StringBuffer* the complexity is reduced to just n :

```

public String statement() {
    final int n = numItems();
    final StringBuffer s = new StringBuffer(n * ESTIMATED_LINE_WI

```

```

DTH);
    for (int i = 0; i < n; i++) {
        s.append(lineForItem(i));
    }
    return s.toString();
}

```



The JLin rule *String Concatenation* will find improper use of the string concatenation operator in your source code.

Creation

It is not necessary to create new string instances explicitly (`new String("text")`), except in some very rare cases where equal strings are considered different when they belong to different objects (the `==` operator is used for comparison).



The JLin rule *String Creation* will find places where strings are explicitly instantiated.

Accessing Individual Characters

The method *substring* extracts arbitrary character sequences from a string. To extract a single character though, the method `charAt()` should be used.



The JLin rule *Substring* will look for method `substring()` where `charAt()` is more appropriate.

Arrays

Arrays have the advantage to be type-safe but have the following disadvantages:

- There is no `contains()` method.
- Arrays do not allow to insert or delete elements.
- Arrays are not embedded in the collections framework.
- Arrays cannot be write-protected.

By the way - type safety can also be achieved by type safe wrappers and by Java Generics in the future.

Example:

```

StringTokenizer tokenizer = new StringTokenizer(poolIDs, " ");
String[] ids = new String[tokenizer.countTokens()];
for (int i = 0; tokenizer.hasMoreTokens(); i++) {
    ids[i] = tokenizer.nextToken();
}

```



```
for(int i = 0; i < ids.length; i++) {  
    addEntryToList(ids[i]);  
}
```

In this example the only purpose of array *ids* is to allow iteration-by-index idiom. Here is an alternative:

```
StringTokenizer tokenizer = new StringTokenizer(poolIDs, " ");  
for (int i = 0; tokenizer.hasMoreTokens(); i++) {  
    addEntryToList(tokenizer.nextToken());  
}
```

Object Class

The root class *Object* represents the basic infrastructure of the Java object model. Descended classes inherit the following methods:

- Identity concept: `equals`
- Support for hash codes: `hashCode`
- String representation: `toString`
- Duplication: `clone`

See book *Effective Java* [\[1\]\[Page 33\]](#) for more details and examples.

The `equals()` Method and Object Identity

The `equals()` method determines if two objects are equivalent. The identity concept is used to distinguish keys in maps and to avoid duplication of elements in sets. Also the containment checks of most utility classes are based on the implementation of the `equals` method.

As a side effect the implementation of the `equals` method is a hot-spot whenever the Java Collection Classes or other container classes are used. Performance is the key for the implementation of `equals`.

Use the following recommendations to obtain a fast implementation of the method:

- Check first for identical objects using `==` and return early.
- Use the hash-code to detect differing objects early (requires fast implementation with cached hash-codes).
- Compare attributes with high chance to differ (broad range) first.
- Compare attributes of basic types (`==`) first.
-



The default implementation of `equals` returns true only for the same instance (`x == y`).

Example:

The following class represents a position in a text (like a bookmark). It will offer superior performance because it will return early for `x.equals(x)` and compares the fields first that

are most likely to differ. In this case rows are more likely to be different because there are usually more rows than columns in a text.

```
public class Position {
    private int _row, _col;
    ...
    public boolean equals(final Object o) {
        if(o == this) { // succeed early
            return true;
        }
        final p = (Position) o;
        return (_row == p._row) && (_col == p._col);
    }
}
```

The hashCode() Method

The hash-code support of Java objects goes hand in hand with the identity concept. There is a logical relationship between the behavior of the *equals* method and the *hashCode* method.



Equal objects have equal hash codes.

For implementations of the *hashCode* and the *equals* method the following holds true:

$\forall x, y$ of object type

- (1) $x.equals(y) \Rightarrow x.hashCode() == y.hashCode()$
- (2) $x.hashCode() != y.hashCode() \Rightarrow !x.equals(y)$

Statement (2) is the conclusion from (1).

The *hashCode* method is often used in classes that manage groups of objects, like the following:

- `HashMap` and `LinkedHashMap`
- `Hashtable`
- `Properties`
- `HashSet`

Hash codes are the quickest way to lookup objects.



The default implementation of *hashCode* corresponds to the default *equals* implementation and therefore returns different values for different object instances.



The *JLin* rule *Equals/Hash Code Test* shows inconsistent implementations.

4.1.2.3 Collection Classes (java.util)

The Java Collection Classes offers a powerful set of facilities to deal with groups of objects. When used consistently in implementations and interfaces they can shorten development time significantly by releasing the programmer from standard tasks like maintaining lists, sorting, removing duplicates and finding objects quickly.

Choosing the Right Implementation Class

The performance of code can be affected significantly by choosing the right collection implementation. Regarding the correct and efficient use of the Java Collection classes, a significant performance setback comes from the low utilization of implementation classes other than `ArrayList` and `HashMap`. These two implementations are used for almost everything, although different implementations would be more appropriate in some situations.

The following rules should be applied to select the appropriate collection class:

- Use `ArrayList` to collect data in order of occurrence (append operation only). Adding or removing elements at the bottom of the list is very fast on `ArrayLists` and `LinkedList`, but the `LinkedList` puts a high load on the garbage collector. This is the most common use case for lists.
- Use `LinkedList` to maintain lists whenever elements have to be inserted or removed at positions *other* than the the bottom of the list. Inserting or removing elements is very slow for `ArrayLists` because at average half of the elements in the list have to be shifted.
- Use `LinkedList` to maintain lists that contain only very few elements and have a lot of instances (for example, 5000 lists with 3 elements each). An `ArrayList` wastes space for unused slots, a `LinkedList` needs more bytes per element.
- Never use `LinkedList` for index-based access. Use only the implementations of `List` that also implement the (tag-)interface `RandomAccess`.
- Use Set types like `HashSet` and `TreeSet` to maintain lists that contain no duplicates (for example, observers). Searching is very quick for sets: $O(\log n)$ instead of $O(n)$. The Set implementation `LinkedHashSet` combines the preservation of the insertion order of the `List` type with the quick lookup of the `HashSet` type.

The values in the table below show the memory requirements of `ArrayList` and `LinkedList` during an append-only operation. The `LinkedList` is faster in filling the list but takes much more time to clean up afterwards. This is due to the complex memory-structure of a linked list (many objects, many references) and therefore a lot of work for the garbage collector.

So the real behavior depends largely on the load that is put on the system. When the CPU load is not high and garbage collections are not frequent, the garbage collection overhead of the `LinkedList` is no problem.

When garbage collection takes place as the CPU is busy the time, the garbage collector needs to work off the memory structure (*GC Full Cycle Time* in the table) will be added completely to the execution time of the application, because the program execution is interrupted until the memory is acquired by the garbage collector.

Appending 10.000.000 elements to a list.

List Class	Memory	Garbage	Execution Time ¹	GC Time ²	GC Full Cycle Time ³
ArrayList	45MB	89MB	620 ms	< 1ms	140 ms
LinkedList	234MB	0MB	460 ms	230 ms	1.550 ms

¹ no garbage collections in the background

² time required to collect the objects in the list after the reference to the list has been cleared

³ time the garbage collector took to traverse the heap for releasable objects (without finding any)

See <http://java.sun.com/developer/JDCTechTips/2002/tt0910.html> for more details on the garbage collection performance.

Java 1.0 Collections (Deprecated)

These classes exist in the early Java versions and have been replaced by the Java Collections Framework as of JDK version 1.2. The new classes are integrated into a framework of types and are not automatically synchronized. The appropriate use of the new classes improves the design and performance.

For all classes that work with JDK 1.2 or later, we do not recommend to use the old classes *Vector*, *Hashtable*, and *Enumeration* for the following reasons:

- The access to the members is always synchronized. This is unnecessary when read-only access is performed concurrently or the context is already synchronized. In other cases it is not sufficient as this type of synchronization does not secure multi-step operations like loops.
- They are considered legacy by the Java creator.
- *Enumeration* behaves unstable when the underlying *Vector* is modified concurrently.
- The naming scheme is inconsistent.
-



Do not use *Vector*, *Hashtable* and *Enumeration*. These are legacy classes which have unsolved issues with performance and scalability in a server environment.

4.1.2.4 I/O Classes (java.io)

File Access

File access from applications must not be used because of performance and security problems. There are several problems related to file access from servlets, portal applications or EJBs:

Developing Well Performing Portal Applications

- Files access is a bottleneck in terms of scalability and performance and is amplified when accessing files on network shares. There is no pooling and no caching available and the number of concurrently open files is usually restricted.
- Files access bares a potential security risk because it has no built-in authorization concept like there is with true J2EE *resources* (for example, JNDI).
- Files are not transactional and therefore only suitable for static information like configuration settings. For these types of data, better facilities are offered by the portal, for example, *profiles*.



The JLin rule *File Access* finds code sections that implement access to files.

Buffering Stream Access

For better performance, byte/character data should always be transferred in chunks. Operating systems use block wise I/O for transfers (files, network), therefore byte-per-byte access will multiply the overhead generated by Java. When byte wise access is required, the use of buffering wrappers (*BufferedWriter*, *BufferedReader*, *BufferedInputStream*, *BufferedOutputStream*) is mandatory.

Releasing Resources

The operating system usually limits the number of I/O related resources, because even inactive resources put load on a system. Therefore the garbage collection mechanisms (finalization) offered by Java are not adequate for the management of operating system resources. In order to prevent shortages it is necessary to explicitly release these resources using dedicated methods (often named *close*).

The following classes represent resources that should be released explicitly:

- *InputStream* (except for *ByteArrayInputStream*)
- *OutputStream* (except for *ByteArrayOutputStream*)
- *Reader* (except for *StringReader*)
- *Writer* (except for *StringWriter*)
- *Connection*

For deeper coverage, see [Resource Management \[Page 33\]](#).



Use JLin Rule *Release of Resources* to automatically detect suspicious code sections.

4.1.2.5 Memory Management

Automatic memory management offered by Java makes development much easier because references do not have to be tracked by the application itself. As soon as an object is not referenced by a variable, it is ready to be reclaimed by the garbage collector built into the JVM. As a result, there is no way to reference objects once the memory has already been

released. This solves the problem of dangling references known in languages with manual memory management.

The problem that arises is that objects that are not used anymore can only be released when there are no longer referenced. Keeping references to unused objects cause memory leaks.

The Java memory management has the following basic rules:

- Every object in the JVM has a counter containing the number of variables that actually refer to it.
- Objects are released by the garbage collector when they are no longer referenced - that is when the counter is zero, because the object cannot be accessed anymore.
- When an object (including class objects) is released by the garbage collector, all references contained in the member variables of the object are discarded too (the reference counter of the referred objects is decremented). When a class is released (because there is no more instance of the class in the system), all static members containing references are also cleared. So further objects can be released by the garbage collection.
- The garbage collector is working asynchronously, in a separate (low-priority) thread. Therefore the instant the memory is reclaimed cannot be predicted as it depends on several factors (system-load, available memory and so on).

There are different garbage collector implementations available and the strategy is much more sophisticated. For more details see:

<http://developers.sun.com/techtopics/mobility/midp/articles/garbage/>

Memory Leaks

Memory leaks are created by keeping a reference to objects that are not in use any more. Often, container objects (like the Java Collection Classes) are the source of such leaks. All objects or classes that are used to collect information are candidates for memory leaks. Especially static variables are dangerous, because the class object itself is not released by the garbage collector. This has to be kept in mind when static lists or maps are introduced.

Example:

Maps (like `HashMap` or `Hashtable`) are used when information *I* has to be associated with an object instance *X*, mapping the key *X* to the value *I*. As soon as the corresponding object *X* is not used anymore, the entry (*X* → *I*) in the map has to be removed, otherwise *X* will stay in the map and the garbage collector recognizes the still existing reference to the map entry. The concept of weak references helps to resolve these issues. For more details see:

<http://java.sun.com/developer/technicalArticles/ALT/RefObj/>

Garbage Collection Slowdowns

In Java every object resides in the heap (in dynamic memory). Java treats all objects by reference. There are no local objects on the stack (compared to local variables of basic type that disappear when their scope is gone). As a result even temporary objects that are used only for a short period of time are subject to dynamic memory management and therefore garbage collection.

This leads to the situation where programs that create a lot of temporary objects put an enormous load on the garbage collection thread. A backlog of garbage objects is building up and the available memory is going down. Eventually the garbage collector has to interrupt the program execution to clean up the backlog. This is called full garbage collection. This kind of emergency reaction is responsible for long delays in the response time, sometimes even more than a minute with no visible progress.

Example of excessive garbage creation (see also [Built-in Types \[Page 33\]](#)):

```
String contents = "";
InputStream is = url.openStream();
int c;
while((c = is.read()) != -1) {
    contents += c; // creates a new object
}
```

For each transferred character a new string is created and assigned to the variable `contents`. For a file that has N characters, $N \times (N/2) \times r$ bytes of garbage memory will be created and has to be reclaimed by the garbage collector.

The JVM option `-verbose:gc` monitors the behavior of the garbage collection and can help to spot memory leaks.

4.1.2.6 Resource Management

Operating system resources like network connections, file handles or database connections are a limited resource. Therefore it is more than good etiquette to release resources as soon as possible. A common misperception is that the garbage collection will take care of that. While this might be true for a single user system with a lot of spare CPU time, the situation in a server environment is much more different.



Release limited resources as soon as possible.

Resources with limited availability should be released as soon as possible and in a reliable way, like calling the release method (for example, `close`) of the resource from inside a finally block. This block has to be declared even when there are no checked exceptions (no try clause).

Because of the high work load typical for application servers, it is more likely that it will take some time for the garbage collector to finalize an object with a resource. The resource is unavailable because to the latency of the garbage collector.

To avoid this situation, all classes that encapsulate limited resources offer methods to release them early.

```
InputStream is = socket.getInputStream();
... // read the file
is.close(); // release resources early
```

While this example works fine for local files it does not handle failures.

Example:

The variable `is` could also point to a network connection that might be interrupted any time - causing an exception. Even when exceptions occur, our resource should still be released before it is discovered by the garbage collector.

```
InputStream is = null;
try {
    is = socket.getInputStream();
    ... // read the file
} catch(IOException ex) {
    ...
    // important
```

```
    } finally {  
        if(is != null) {  
            is.close();  
        }  
    }
```

4.1.3 Portal Application Programming Model

The portal application programming model describes the life cycle and the execution state of a portal application.

4.1.3.1 Lifecycle of an Portal Application

The lifecycle of a portal application is important to perform the initialization and clean up of portal components in an efficient and reliable way. It is composed of three steps:

Initialization

Operation

Disposal

The implementation of these steps depends mainly on the technology chosen for the portal component.

See the *PDK:Java Developer/Documentation* for more details.

Initialization

Every portal application implementation has an initialization body. It should contain the coding to render the component on the Web client.



Delay complex operations.

Complex parts of the portal application that are not used in the initial view should not be initialized before the user demands it.

Example:

A portal application has a user interface with tab strip that has two tabs. The first tab contains a clock and the current date. The second tab contains the appointments for the user retrieved from a backend system.

The initialization should contain the steps to render the first tab. Build the tab control and initialize the widget for the clock. For the second tab it just checks the configuration for the connection parameter to connect to the backend system containing the calendar application.

The initialization should not retrieve the appointment data necessary to render the second tab. This should be done when the user actually requests this tab.

Portal Application Models Initialization

The method called for initialization depends on the super class belonging to the portal application model. The methods to implement depend on the selected programming model, so check the documentation for details.

Initialization in portal application models

Model	Basetype	Initialize
Handcrafted portal application	IPortalComponent	constructor of the implementing Java class (IPortalComponentInitContext)
Standard portal application	AbstractPortalComponent	init(IPortalComponentIC)
HTMLB portal application	DynPage/JSPDynPage (PageProcessorComponent)	doInitialization()

Operation

In this phase the portal component is initialized and ready to handle requests. Steps that are not handled in the initialization phase are handled now.

The major objectives for the *Operation* phase are:

- Proper isolation of users and sessions.
- Limitation of resource requirements.
- Stability and verbose error management.

The methods to implement depend on the selected programming model, so check the documentation for details.

Initialization in portal application models

Model	Basetype	Callback
Handcrafted portal application	IPortalComponent	Service (IPortalComponentRequest, IPortalComponentResponse)
Standard portal application	AbstractPortalComponent	on...(...)
HTMLB portal application	DynPage/JSPDynPage (PageProcessorComponent)	doProcessBeforeOutput(), doProcessAfterInput()

HTMLB portal applications

HTMLB portal applications using JSP implement the class *JSPDynPage*, otherwise the *DynPage* class has to be implemented.

For more details see:

<http://<pdk-server>:<port>/irj/portalapps/com.sap.portal.pdk.htmlb.htmlbmanuals/docs/dynpage-01.html>.

Disposal

It is strongly recommended that portal components when shutdown (removed from the container) release all used resources, especially caches and connections to backend systems and databases.

All threads created by a portal application, although not recommended, have to be stopped.

Disposal in portal application models

Model	Basetype	Callback
Handcrafted portal application	IPortalComponent	destroy()
Standard portal application	AbstractPortalComponent	destroy()
HTMLB portal application	DynPage/JSPDynPage (PageProcessorComponent)	destroy() – inherited from Page ProcessorComponent. POM triggers cleanup when portal application is removed from POM tree.

4.1.3.2 Storing Execution State

The portal content is handled *stateless*. An portal application has to keep track of its current execution state that includes, for example, the status of data handling. The portal works like a servlet. It uses a single instance to handle all requests from different users and different sessions. Therefore the current execution state cannot be kept in the portal application implementation class.

The following document contains detailed information about the different contexts to store data and the lifecycle:

<http://<pdg-server>:<port>/irj/portalapps/com.sap.portal.pdk.htmlb.htmlbmanuals/docs/usageofbeans.html>

Component Context

The component context can be used for application relevant data but it is not recommended. Data kept in the component object is not very secure. The component context can be released anytime by the container and replaced by another instance.

Request Context

Data stored in the request context are available as long as the request object exists. It can be used to communicate between different components.

The POM node associated with the request object can be used to store custom attributes and can be accessed with the method `getNode()`.

Access to request context in portal application models

Basetype	Access to IPortalComponentRequest
IPortalComponent	The request object is passed as parameter to the <code>service</code> method.
AbstractPortalComponent	The request object is passed as parameter to the <code>do...</code> methods.
DynPage/JSPDynPage (PageProcessorComponent)	In the (JSP)DynPage class, the request object is returned by method <code>getRequest()</code> .

Session Context

The session context can be used to for user specific data. The data is available as long as the browser session exists. It is accessible over the request object using the method `getComponentSession()`.

State Management in JSP-based portal applications

The execution state in JSP-based portal components is usually handled with Java bean objects that are embedded into the page with the `jsp:useBean` tag. The lifecycle of these objects is managed by the JSP container according to the scope declared for the bean.

State management in portal application models

Scope	Visibility and Lifetime
application	The data is shared by all instances of the portal application. The lifecycle corresponds to that of the deployed portal application.
page	The data is shared by all portal applications contained in the same page.
session	The data is shared by all portal applications that execute in the browser session.
user	The data is user specific. It is visible to all portal application instances used by the user, also for several sessions.

Data stored in the *user* or *application* space have a long lifetime. Incomplete clean up procedures can have a significant impact on the performance of the overall system and can create memory leaks.

4.1.3.3 Portal Application Performance

Observed and Real Performance

Performance can be into performance sensed by the users and performance from a technical point of view.

Performance sensed by the user is difficult to judge. Several psychological factors come into play. The most important performance issue for user is the delay between a user action and the response.

Examples:

- Time from the start of an application until it is ready to use.
- Rendering time of a Web page in a browser.
- Time to load a pop up menu to select user input.

If an operation is more time consuming the user should be informed as follows:

Developing Well Performing Portal Applications

- What operation is currently in progress.
- Progress bar.
- Percentage already processed.
- Estimated time when the task is finished.

Portal applications using these techniques are usually rated “quick” by the user, although it is slow from the technical point of view.

When portal applications need information from a backend system it is important that the applied query is efficient and fast and that the application avoids querying the same information several times.

Summary:

Technical performance can be improved with efficient algorithms, fast connections and clever caching.

Caching

The term *caching* refers to all strategies that hold recently-retrieved data to avoid the overhead retrieving the data again. For portal applications caching is used then accessing backend systems.

Example:

A portal application displays the current temperature that is provided by the commercial weather-service *Cats-and-dogs.com* based on *WebService* technology. The portal application will update the temperature in a five minute interval. Since the temperature is independent from the username, the values will be queried only once from the weather service and stored for all users in the portal. The portal application has to do the following:

1. Keep the weather values in an area that is shared by all users on the node: portal application class.
2. The weather values have to be accessible for five minutes.

Because of the cache the weather-service is called every five minutes and not for every user that accesses the weather portal application.

Areas relevant to caching are:

Scope: User, role, group, application, area, country

Expiration: fixed interval, time-based, never

Survival criteria: size, usage frequency, last usage

Representation: raw data, business-level data, presentation-level data

Following data sources should be cached:

- Databases
- Connected SAP backend systems
- File I/O
- Any kind of data that is retrieved through network connections.

The representation of the data in a cache is also important. For the weather portal application example can be implemented with three different cache levels:

1. The lowest level would only cache the raw data obtained from the weather service which is the XML document containing the response to the most recent query. The advantage is that the entire message is stored and no information is lost. The

disadvantage is that the needed data entry has to be extracted every time when the temperature value has to be refreshed.

2. On the business level the object containing the weather data in a normalized form is cached.
3. On the presentation level the weather data is cached in the form it is displayed. If the portal application displays the data in a common layout, the entire HTML page with the current business object state can be cached, instead of generating the HTML page out of the business level data again and again.

Caching Scopes

As described above the *SAP Enterprise Portal* offers various places to establish a cache (for example, request or session context). Generally speaking, a cache context should be as wide as possible but it must consider the following logical and performance issues:

4. What is the cache eviction (or timeout) policy and how is outdated data handled (prefer a quick check over notification patterns).
5. A cache must never grow unlimited! The implementation of the cache has to limit the number of objects / entries kept in the cache. Memory is a very valuable resource that is needed to process requests and not to store data that has already been accessed once before.

Deferred Initialization

The rendering time of a portal page is a big issue. A portal page often contains several portal applications. The user will most likely not use all displayed portal applications. Therefore it is very important that all portal applications used on a portal page initialize and render fast. The initial display of an portal application should not collect extensive data. This should be done on demand. This will improve the overall speed of page rendering for the majority of users.

Portal Applications on the Start Page

The start page is loaded right after the user has logged on to the portal. Because the start page is displayed quite frequently it must have a good performance. Following rules apply for the start page:

- Data initialization has to be reduced to the absolute minimum.
- The start page should have a lean layout to speed up the initial rendering.
- The memory footprint must be as low as possible (not more than 1 MB) because there will be several active instances at once.

4.1.4 Enterprise Portal Services

Some Enterprise Portal services do need more resources and can influence the performance.

4.1.4.1 JCo Client Service

Portal applications that use the JCo-API can create unexpected spikes in the memory consumption. Especially the method `JCo.createRepository()` creates a lot of temporary

Developing Well Performing Portal Applications

objects causing increased garbage collection overhead and even *OutOfMemory* exceptions when used carelessly. To minimize the usage of this expensive operation, the caching-facilities introduced with the latest JCo Client-Service should be used.



JCo is deprecated.

Please note that JCo has been replaced by the SAP Connector Framework (based on JCA) as of EP 6.0. It is only included for compatibility reasons. For new developments Connector Frame/JCA must be used.

The call of method `JCO.createRepository(0)` should be replaced by the method `IJCOClietService.getRepository()`.

Example of a not recommended implementation:

```
String sysId = myProfile.getProperty("SystemIdentifier");
IJCOClietService clietService = (IJCOClietService)
    PortalRuntime.getRuntimeResources.getService(IJCOClietService.K
EY);
IJCOClietPoolEntry jcoPoolEntry =
    clietService.getJCOClietPoolEntry(sysId, componentRequest);
...
JCO.Client client = jcoPoolEntry.getJCOCliet();
IRepository repos = client.createRepository("mydestination", clien
t);
...
client.execute(...);
...
jcoPoolEntry.release();
```

Example of recommended implementation:

```
String sysId = myProfile.getProperty("SystemIdentifier");
IJCOClietService clietService = (IJCOClietService)
    PortalRuntime.getRuntimeResources.getService(IJCOClietService.K
EY);
IJCOClietPoolEntry jcoPoolEntry =
    clietService.getJCOClietPoolEntry(sysId, componentRequest);
...
JCO.Client client = jcoPoolEntry.getJCOCliet();
IRepository repos = clietService.getRepository("mydestination",
    componentRequest);
...
client.execute(...);
...
jcoPoolEntry.release();
```

An introduction to the JCo client service is available on the SDN at:

<https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/com.sapportals.km.docs/documents/a1-8-4/JCo%20Client%20Service>

4.1.4.2 User Management Engine (UME)

The user management engine (UME) offers a comprehensive API to manage user, roles, groups and the associates access control lists (ACLs). The implementation is based on the Portal Content Directory (PCD) and inherits some of its characteristics.

The UME API works with persisted data and the data has to be consistent on different portal nodes. This makes calls to the UME API rather complex. Performance often depends on the structure that is used to represent the data and is also affected by the cluster-communication to synchronize the collaborating nodes.

Checking for Existence of UME Entities

Currently the UME API has no method to check if a user, role or a group is known in the system. The straightforward workaround to get the corresponding *IUser*, *IRole* or *IGroup* object cannot be recommended because this will also load the associated attributes.

Recommended code to search for a user:

```
boolean existsUserByUniqueName(final String uname) throws UMException {
    final IUserFactory userFactory = UMFactory.getUserFactory();
    final IUserSearchFilter filter = userFactory.getUserSearchFilter();
    filter.setUniqueName(uname, ISearchAttribute.EQUALS_OPERATOR, false);
    return userFactory.searchUsers(filter).hasNext();
}
```

The code also works for roles and groups simply by replacing the `searchUsers()` method to `searchGroup()` or `searchRole()`.

User-Role Assignment

The internal representation of the assignment between users and groups has a significant influence on the performance of the UME. User and role are in a bidirectional relationship and can be associated in two directions. Due to the internal data management of the UME, links from the role to the user (add user to role) are handled faster when multiple users have to be processed.

Example of a not recommended implementation:

```
UMFactory rf = UMFactory.getRoleFactory();
rf.addUserToRole("petermueller", "role1");
rf.addUserToRole("annefranklin", "role1");
rf.addUserToRole("norasmith", "role1");
```

Improve performance and save memory with this approach:

```
UMFactory rf = UMFactory.getRoleFactory();
IRole role;
role = rf.getMutableRole("role1");
role.add("petermueller");
role.add("norasmith");
role.add("annefranklin");
role.save();
role.commit();
```

See **SAP Note 746682** (*Performance problems when assigning roles/groups via UME API*).

User-Group Assignment

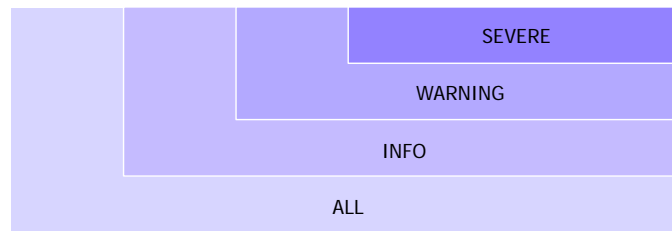
User group assignment is similar to user-role assignment.

See **SAP Note 746682**.

4.1.4.3 Logging

Logging must be implemented with the PRT Logging API. Printing to the standard output streams will slow down performance and the information is hardly noticed by the administrator. The logging API of the portal generates diagnostic information in an efficient way and stores the information in a database for easy retrieval.

The preparation of the logging messages can take some time to collect and format the information. Avoid unnecessary overhead by checking the log level before you request additional information. Benefit from the hierarchy of the log levels (inclusion relationship).



The levels can be described in a mathematical formula as follows:

$$\text{SEVERE} \subset \text{WARNING} \subset \text{INFO} \subset \text{ALL}.$$

For a Java program this can be translated into the integer value associated with the level:

```
SEVERE.intValue() < WARNING.intValue() < INFO.intValue() < ALL.intValue()
```

A special level is the pseudo-level *OFF* (not in the picture) that indicates that logging is turned off. It has the lowest integer value, so no extra checking is required. Therefore a message of level *m* is logged when the configured logging level *L* is greater or equal: $L \leq m$.

```
ILogger lg = PortalRuntime.getLogger(); // get default logger
// check if the logger is configured to include INFO messages
if (lg.getLevel().intValue() >= Level.INFO.intValue()) {
    String hostname = resolveHostname(myAddr);
    lg.info("connecting to: " + hostname);
}
```

4.1.5 Database Access

Managing Connections

Portal applications that have to access databases directly should not try to create connections themselves. Portal application must use the built-in connection pools in order to limit the number of concurrent connections and to prevent possible low resources that would affect other applications.

The *SAP Connector Framework* comes with a JDBC adapter that can be used to manage databases like regular J2EE *resources* under control of the application server. This will positively affect the server performance and stability.

A good approach would be to extract the database logic from the portal application and put it into an Enterprise Java Bean (EJB) or at least a portal service, using portal applications only as front-end components. EJBs offer a superior infrastructure to wrap backend systems. The separation of front-end and business logic is the best solution to increase performance and

reduce maintenance. Also business logic wrapped in an EJB is easier to reuse by other application and can easily be published as *Web Service*.

4.1.5.1 Releasing JDBC Objects

It is good practice to release database resources as soon as possible because the number of database connections is limited and active statements and result sets need memory. If you do not close resources explicitly they may be blocked for a longer period which leads to a bottleneck when the load on the server becomes higher.

See also [Resource Management \[Page 33\]](#) for general rules about resource management.

4.1.5.2 Transactions

A single logical operation can result in several operation for the database. The transaction guarantees that the database stays consistent during these operations. A transaction contains several database operations that are either executed all at once or not at all.

The transaction ensures the following:

- Atomicity: The operations are executed all at once, or not at all.
- Consistency: The affected data is kept in consistent state.
- Isolation: The software is protected from external manipulations to shared data (the degree may vary though).
- Durability: Once a transaction has been confirmed, the result is stored and will not be reversed.

Transactions also improve the performance.

```
Connection conn = ... // get connection from pool
conn.setAutoCommit(false);
... // do several updates
conn.commit();
```

4.1.5.3 Harnessing SQL

Relational databases and the SQL language offer superior capabilities to filter large amounts of information. The SQL interpreter is directly located at the data source, so database operations are executed much faster than database operations from a connected application, like a Java program using JDBC.

In order to improve execution times and save memory it is mandatory to delegate every possible processing work to the database system. Use the following SQL features to improve the performance significantly:

- Joins
- Sub queries
- Sorting
- Column functions

Keep in mind that complex SQL statements may be demanding for the database. For more details and background information on database performance review the SAP help at:

Developing Well Performing Portal Applications

<http://help.sap.com/> and search for the terms:

"SQL Performance", "Database Performance" or "Open SQL".

Example of a not recommended implementation:

A Java program scans the entire table to find out when a user performed his latest activity.

```
String query = "SELECT * from ActivityLog WHERE User = '"
    + value + "'";
long timestamp = 0;
ResultSet rs = _dbAccess.query(query);
while (rs.next()) {
    timestamp = rs.getLong("TIMESTAMP");
}
return timestamp;
```

The example has following flaws:

- The database is queried to return all columns (SELECT *) although only the TIMESTAMP column is needed.
- A Java iteration instead of the column functions of SQL that iterates over an entire column is used, to find the latest entry.

Example of a recommended implementation:

```
String query = "SELECT MAX(TIMESTAMP) FROM ActivityLog WHERE User = '"
    + value + "'";
ResultSet rs = null;
try {
    rs = _dbAccess.query(query);
    if(rs.next()) {
        return rs.getLong(1);
    } else {
        return -1;
    }
} finally { // see section Resource Management
    if(rs != null) {
        rs.close();
    }
}
```

The columns referenced by the *WHERE* or the *ORDER BY* clause should have an index.

4.1.5.4 Prepared Statements

To invoke an SQL statement with different parameters, JDBC offers the prepared statements feature. Prepared statements are SQL statements that contain question marks for every parameter. The SQL statement has to be precompiled by the DBMS and on every execution the current values are filled in. Prepared statements have the ability to insert also values that cannot be expressed in SQL (as string), like BLOBS.

Example without prepared statements:

```
Collection newMembers = ...;
```

Developing Well Performing Portal Applications

```

Connection conn = ...;
Statement stmt = conn.createStatement();
for(final Iterator i = newMembers.iterator(); i.hasNext();) {
    Member m = (Member) i.next();
    stmt.executeUpdate("INSERT INTO Members VALUES ('"
        +m.getName()+"', '"
        +m.getAddress()+"', '"
        +m.getCity()+"'");
}
stmt.close();

```

The example has following flaws:

- A new string is created in the loop.
- Performance is lost because the database does not know that there are more statements of the same structure.

Example of recommended implementation:

By applying prepared statements we can save the entire string processing and our database (or driver) can perform the operations quicker. The best solution is to prepare the statement once and not each time before the execution.

```

Collection newMembers = ...;
Connection conn = ...;
PreparedStatement stmt = conn.prepareStatement
    ("INSERT INTO Members VALUES (?, ?, ?)");
for(final Iterator i = newMembers.iterator(); i.hasNext();) {
    Member m = (Member) i.next();
    stmt.setString(1, m.getName());
    stmt.setString(2, m.getAddress());
    stmt.setString(3, m.getCity());
    stmt.executeUpdate();
}
stmt.close();

```



The *JLin* rule *Candidates for PreparedStatements* checks where prepared statements could be used.

4.1.5.5 Stored Procedures

Stored procedures are written entirely in SQL and are executed on the database system.



Stored procedures are not recommended by SAP.

Stored procedures are not recommended by SAP. The reason is that stored procedures put the load on the database, the central resource, and therefore restricts the scalability of the whole cluster / system.

Stored procedures can only be considered as quick correction of an architecture / database design problem in a customer project that uses one database.

4.1.6 Enterprise Portal Performance Ruleset for JLin

The rules in this document for the static code analysis plug-in *JLin Autocheck* are available on the *SAP SDN*. They can be installed in the *SAP NetWeaver Developer Studio* by extracting the ZIP file `EP_JLin_Rules.zip` to the root folder of the JDT installation directory, for example,

C:\Program Files\SAP\JDT

After restarting the *SAP NetWeaver Developer Studio*, the *JLin* settings will show a new set of rules called `eptf tests`. Activate all tests in this rule set by selecting the check-mark. Now you can run a static code analysis by invoking the menu entry as follows:

Run → Run As → JLin.

The EP Taskforce rule set

Rule	Short Description
Usage of Threads	Indicates creation of new threads. This is forbidden for portal application.
File access	Indicates file I/O. This is a bad practice and has critical performance impact when performed on every request.
Iteration over Map entries	Marks inefficient idioms for map iteration.
String concatenation	Alerts bad usage of the + operator for strings.
Prepared Statements	Indicates non-prepared SQL statements in a loop.
Release of resources	Looks for coding where limited resources are not released appropriately.
Unnecessary creation of String objects	Looks for coding where strings are explicitly instantiated using <code>new String(...)</code>
Unnecessary sub strings	Looks for the usage of method <code>substring</code> for the extraction of individual characters.
NullPointerException Check	Indicates where <i>NullPointerExceptions</i> are caught to avoid null-checking.
Usage of obsolete collections	Marks places where obsolete collection classes <i>Vector</i> , <i>Hashtable</i> , <i>Enumeration</i> are used.
Loop condition	Looks for repeated evaluation of constant expressions in loop conditions. For example, <code>i < size()</code> .

4.1.7 Checklist for Reviews

This section contains a check list for a step by step code review.

Prepare Project in NetWeaver Developer Studio

- Create a project in the *SAP NetWeaver Developer Studio* to improve the efficiency of the code review.
- Configure and use *JLin* (see section [Enterprise Portal Performance Ruleset for JLin \[Page 33\]](#)) to get a list of suspicious coding that should be processed with priority.

Database

- Look for non-prepared statements and evaluate according to section Stored Procedures.
- Check SQL statements for efficiency
- See if connections are released properly and result sets are closed soon.

4.2 File Access

- Look for file access happening in the application (*File*, *FileInputStream*, *FileOutputStream*).

4.2.1 Ensuring Supportability with Metrics and Audits

Maintainability is the key issue for the long-term success of software projects and is an important part of the quality process in software projects. Software metrics and audits capture typical shortcomings in the field of error handling and complexity. So it is often necessary to reduce consolidate the current version before further changes are made. This process is called *Refactoring* [\[6\] \[Page 33\]](#).

JLin is the primary tool for *SAP NetWeaver* developers to ensure the supportability and robustness of the program code while writing it. *JLin* comes with plenty of rules that help finding potential risks for maintenance and robustness in the program code. Unfortunately these rules have to be configured by hand to be really effective.



Use strict settings to enforce maintainability right from the beginning. Static code analysis is a powerful quality assurance tool especially when working with freelancing software developers.

The following tools can be used to generate metrics and audits from source code.

- *JLin*: Is supported in the *SAP NetWeaver Developer Studio*.
- *PMD*: Open-source plug-in that can be used to detect *copy&paste* coding. Not available in the *SAP NetWeaver Developer Studio* – can be installed separately.
- *JDepend*: Open-source plug-in that can be used to create design metrics and detect cyclic dependencies between packages. Not available in the *SAP NetWeaver Developer Studio* – can be installed separately.



In addition to metrics, every component should be profiled before it is used productively. For every click and scenario measure the following:

File Access

- How much CPU is used?
- How much memory (garbage) is used?
- Are there memory leaks?
-
-

Metrics

Lines of Code (LoC)

The LoC is the sum of all code lines in a compilation unit or class. For Java, empty lines, Javadoc and comments are not counted. Methods with a high LoC count will most likely cause problems if they have to be changed, fixed, or extended.

Cyclomatic Complexity (CC)

The Cyclomatic Complexity measures the program complexity. There is a strong connection between this indicator and error density. Cyclomatic Complexity is defined as the number of decisions (*if*, *for*, *while* and *do* statements) per method or function, + 1. Switch statements do not increase the Cyclomatic Complexity in this definition so a method or function has a minimum CC of 1.

A CC value in the range of 10-15 should be considered as maximum. Values below 7 are satisfactory, below 5 would be good.

Methods with a CC over 15 should be *refactored* following the *Extract Method* pattern before the next change or enhancement. For more details see:

<http://www.refactoring.com/catalog/extractMethod.html>



Refactoring in the SAP Netweaver Developer Studio

The *SAP Netweaver Developer Studio* contains *Refactoring* tools that can be used to safely break up complex methods or classes into smaller entities. Keeping software manageable is no magic with these tools and a catalogue of patterns for *refactoring*.

For more details see: <http://www.refactoring.com/>

A practical use case is described at:

<http://www.onjava.com/pub/a/onjava/2004/06/16/ccunittest.html>



The *JLin* rule *Cyclomatic Complexity Metric* can check for thresholds. Before using CC this rule should be reconfigured with `INFO_LEVEL=7`, `WARNING_LEVEL=16` and `ERROR_LEVEL=30`.

File Access

The disadvantage of CC is that it counts conditional and iteration statements only. So a method with 200 statements and method calls has a CC of 1. For more information on CC please have a look at the following Web page:

<http://www.sei.cmu.edu/str/descriptions/cyclomatic.html>.

Depth of Nesting

This test procedure counts how deep if, switch, for, while, and do statements in a constructor or method are nested. Every loop-nesting level means another level of complexity. A method of complexity $O(n)$ explodes to complexity $O(n^3)$ when called in two nested loops and becomes a performance problem. A value of 3 should be considered as maximum.



The *JLin* rule *Max nesting level per method* can check for deeply nested methods. Before using this test this rule should be reconfigured with `INFO_LEVEL=4`, `WARNING_LEVEL=6` and `ERROR_LEVEL=50`.

If a the depth of nesting value is very high it could be necessary to refactor the code according to *Replace Conditional by Polymorphism*. For more details see:

<http://www.refactoring.com/catalog/replaceConditionalWithPolymorphism.html>

Number of Parameters

This test procedure counts the parameters of a constructor or method. A value of 7 should be considered as maximum.



The *JLin* rule *Arguments per method* checks the number of parameters for a method or constructor. Before using this test this rule should be reconfigured with `INFO_LEVEL=8`, `WARNING_LEVEL=11` and `ERROR_LEVEL=20`.

Number of Attributes

Classes with a very high number of attributes often represent a flat composition of several entities. It is recommended to extract groups of related attributes and their methods to classes and use composition instead.

Number of Operations

If a class has a high number of operations, it should be considered to split up the method.

File Access



The *JLin* rule *Methods per Class* checks classes that are candidates for decomposition. Before using this test the rule should be reconfigured with `INFO_LEVEL=30`, `WARNING_LEVEL=50` and `ERROR_LEVEL=100`.

For more details see:

<http://www.refactoring.com/catalog/extractClass.html>.

Audits

Audits apply pattern matching to Java source code to find common mistakes and maintenance flaws.

Copy & Paste Programming

Copy and paste is a popular way to reuse existing functionality in a different context. Unlike a method, class, or package, there is no formal interface (signature, scope, type) between the pasted code and its environment, so copy and paste is pretty unsafe.

Problems of copy & paste:

- **Change Management**
Copying program code also copies the bugs. To keep track of the locations where the duplicated code has been used is quite impossible.
- **Productivity**
Changes made to any part of the code must be applied to the other methods where the code has been copied to.
- **Stability**
Side effects may break "proven code" because the environment is different. This applies especially to copied code snippets.
- **Reusability**
Differences and similarities of replicated code sections are difficult to find because they are hidden in the code and sometimes slightly altered. Copy and paste also prevents reuse in dependent parts of a program because it wraps common behaviour in different interfaces.
- **Overall design**
New classes can be created very fast with copy and paste and can lead into spending less effort on good and reusable design and results in long and complex methods.

For more details see:

http://www.onjava.com/pub/a/onjava/2003/03/12/pmd_cpd.html

This is certainly a very controversial topic. For more details refer to the book [Anti Patterns \[Page 33\]](#).

Cyclic Dependencies between Packages

For more details see:

<http://www.clarkware.com/software/JDepend.html>

4.2.2 References/Bibliography

	Author	Title	Publisher	Year
[1]	Jochua Bloch	Effective Java	Addison-Wesley	2001
[2]	Jack Shirazi	Java Performance Tuning – 2nd Edition http://www.javaperformancetuning.com	O' Reilly	2003
[3]	William J. Brown	Anti Patterns	John Wiley & Sons	1998
[4]	Glen McCluskey	Using ArrayList and LinkedList	Sun Microsystems	2002
[5]	Arthur H. Watson, Thomas J. McCabe	Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric	NIST - National Institute of Standards and Technology	2000
[6]	Martin Fowler	Refactoring – Improving the Design of Existing Software	Addision-Wesley	1999

4.3 General Rules and Guidelines for Managing Exceptions

Exception handling influences the quality of Java programs because:

- It ensures that the program can continue in error conditions.
- It allows the user to find the problem, which caused a thread to terminate. An exception that caused the termination of a portal application should lead to the causing problem by giving meaningful information.

Coding Rules

Rule	Error Example	Explanation
Always pass the original exception	<pre>catch (IOException e) { throw new RuntimeException("Problem in I/O."); }</pre>	<p>The original problem may be hidden if you do not do this. The original exception can be passed on by adding it to the message:</p> <pre>("message: " + e);</pre> <p>It is better to use a constructor that can take an exception as argument as well, for example:</p> <pre>new PortalRuntimeException("Error</pre>

General Rules and Guidelines for Managing Exceptions

		in I/O.", e)
Always include a message	<pre>throw new IOException();</pre>	<p>There is always additional information which is useful to the user. If possible this information should be constructed dynamically. However, this should not introduce problems. The following example produces a <i>NullPointerException</i> if the context is null:</p> <pre>throw new RuntimeException("Error in "+ context.getComponent());</pre>
Do not make wrong assumptions	<pre>catch (NamingException e) { throw new PortalRuntimeException("Reg istry not found:" + e); }</pre>	There are many subclasses of <i>NamingException</i> and the reason may be other
Do not have empty exception handlers	<pre>catch (Exception e) { // ignore }</pre>	
Do not only print the stack trace	<pre>catch (NamingException e) { e.printStackTrace(); }</pre>	<p>If <i>stdout</i> is not redirected, the exception is only visible on the console. The console, however, may not always be there or accessible, for example the application was started using <i>javaw</i> which does not open a console. In addition, there is sometimes too much output on the console thus making it impossible to follow the output. The exception should be logged.</p>
Log important Exception	<pre>catch (IOException e) { throw new RuntimeException ("Exception while reading service parameters: " + e); }</pre>	<p>In general, when important information is lost in an exception handler because it is not passed on, it should be logged. The following line logs the stack trace of the exception above:</p> <pre>PortalRuntime.getLogger().seve re(e, " Exception while reading service parameters.");</pre>

5 Reference

This section contains the following:

- [APIs \[Page 33\]](#): Javadocs for portal and portal runtime APIs

Portal APIs

- [Samples \[Page 33\]](#): Sample code for portal applications

5.1 Portal APIs

The Javadocs for the portal and portal runtime APIs are located on the SAP Developer Network (SDN) at <https://www.sdn.sap.com/irj/sdn/javadocs>.

5.2 Samples

Code samples for portal applications are included within the NetWeaver Developer Cockpit, a portal business package that can be downloaded from the SAP Developer Network (SDN).

For more information, go to the Downloads section at <http://sdn.sap.com>.