How to Use Web Dynpro for Java Components and Component Interfaces

Applies to:
Web Dynpro for Java 7.11. For more information, visit the Web Dynpro Java homepage.

Summary
In this tutorial you will learn how to use Web Dynpro Components and Component Interfaces and selected new 7.1 features.

Author: Web Dynpro Java Team
Company: SAP AG
Created on: 29 June 2010
Table of Contents
Prerequisites.................................................................................................................................3
Objectives.......................................................................................................................................3
About Components and Component Interfaces.............................................................................4
The Tutorial Application..................................................................................................................4
  Overview .....................................................................................................................................6
  Component Usages and Lifecycle Control of Component instances............................................7
  Component Interface Inheritance ..................................................................................................8
  View Containers in Interface Views ..............................................................................................10
  Referencing Mode ........................................................................................................................12
  Component Query Service ............................................................................................................13
  Dynamic Component Embedding .................................................................................................15
Componentization ..........................................................................................................................19
Further Information ........................................................................................................................20
Legend 21
  Text Symbols .............................................................................................................................21
Copyright .........................................................................................................................................22
Prerequisites
You need to install the SAP NetWeaver Developer Studio (NWDS, version 7.11 or later) to compile and deploy the tutorial application. The SAP Java AS to which this application is deployed must be of the same version as the NWDS or a newer version.

The tutorial application is available as a development component (DC). You need to import the software component HM-WDUIDMKTCNT, which contains the DC tc/wd/tut/comp/adv. The exact steps are described in a separate document.

Objectives
After reading this tutorial, you will know how to …
- … use Web Dynpro Components and Component Interfaces.
- … use View Containers in Interface Views to implement layouts.
- … share a Component instance using the Referencing Mode.
- … dynamically embed an Interface View of a used Component.
About Components and Component Interfaces

For a general overview of Web Dynpro Java Components and all their facets see the provided links in the Further Information section especially the NewWDJavaComponenModel_NWCE71 document.

The Tutorial Application

The tutorial application provides different views on data of employees of an imaginary Company iTelO such as the name, addresses and department.

When the application is started the following screen appears; it consists of two parts.

At the top of the screen two viewers are displayed. One provides an overview of all employees and the second provides more details about the selected employee. The user can use the buttons at the top to change the layout and influence how these two viewers are arranged.
At the bottom of the screen a toolbar provides list of all available viewers on the employee data. The user can select the viewers and can choose *Add* to add them to the UI.

The following section provides a short overview of the project structure (Components and Component Interfaces). The subsequent sections focus on different details of the implementation of the tutorial application.
Overview

The tutorial project contains several Component Interfaces and Components which are briefly introduced in this section.

The Component EmplData is the data provider for the tutorial application. It reads the necessary information about the employees and provides this information to other Components via its Context. The Component EmplData implements the Component Interface IEmployeeData and is an example for a faceless Component because it does not provide any visual parts.

The Component Interface ILayout defines a layout with two placeholders for external UIs. The implementations of this interface are the Components ColLayout and RowLayout which each arrange the placeholders in a different way.

The Component Interface IViewer is the common interface for all Components that visualize the employee data. The following Components implement this interface and visualize different parts of the employee data: EmplOverview, EmplAddr, EmplDetail and EmplPic.

To be able to show the Component Interface Inheritance feature as well, the Component Interface IInfo is introduced: this extended by the IViewer Component Interface.

The Tutorial Component is the Root Component of the application and provides the first part of the UI. The Dynamic Component is responsible for the second dynamic part of the UI.
Component Usages and Lifecycle Control of Component instances

ComponentUsages can either point to a Component Interface or to a Local Component Interface (equivalent to the Component).

In the latter case the Component that should be used is uniquely identified and therefore an instance can be created without providing explicit identification information during creation time.

The createOnDemand state means that the Web Dynpro framework automatically instantiates the Component used when it is accessed for the first time (for example calling a method or displaying an Interface View on the UI, …). You do not have to take care about the life cycle of the Component. An example for such an ComponentUsage is the usage of the Dynamic Component in the Tutorial Component.

The manual state allows a more fine-grained handling of the lifecycle of the Component used. This state is compulsory when the Component Usage points to a Component Interface because in that case the concrete implementation of the interface must be provided during the initialization of the Component used. Most of the ComponentUsages in the tutorial application are of this type.

The switchLayout() method of the Tutorial Component shows the usage of the lifecycle control methods createComponent(), deleteComponent() and hasActiveComponent() of the IWDComponentUsage interface. This method is called when the layout on the UI should be toggled between a row layout and a column layout. Technically the implementation of the ILayout Component Interface used is changed. First the currently used layout Component instance is destroyed and then the new layout Component is initialized.
(Note: In the tutorial application all names of Components and Component Interfaces which are used in the Code are defined as constants in the Java Class Constants).

```java
public void switchLayout() {
    //@@begin switchLayout()
    IWDComponentUsage layoutUsage = wdThis.wdGetLayoutComponentUsage();
    //Delete the active used component if existing
    if(layoutUsage.hasActiveComponent()){
        layoutUsage.deleteComponent();
    }
    //Initialize the new layout component depending on the selection.
    String compName =
        wdContext.currentContextElement().getColLayoutUsed() ? Constants.COL_LAYOUT_COMPONENT_NAME :
            Constants.ROW_LAYOUT_COMPONENT_NAME;
    layoutUsage.createComponent(compName, Constants.DC_NAME);
    //@@end
}
```

In addition to the initialization of a new Component instance on a “manual” Component Usage, there is also the option of referencing another Component Usage instead. This is described in the Referencing Mode section below.

**Component Interface Inheritance**

As of SAP NetWeaver 7.1* it is possible that a Local Component Interface or a Component Interface extends another Component Interface.

As an example the tutorial project contains a Component Interface IInfo which defines a getInformation() method. This method should return a short description of what the implementing Component is doing. The Component Interface IViewer extends this interface.

So each Component that implements the IViewer interface promises to also implement the IInfo interface. The following code is taken from the EmplOverview Component as an example.

```java
//@@begin javadoc:getEmployeeDataUsage()
/**
 * Implementation of IViewer Component Interface.
 * @return The usage of this to the employee data provider.
 */
//@@end

public IWDComponentUsage getEmployeeDataUsage() {
    //@@begin getEmployeeDataUsage()
    return wdThis.wdGetEmployeeDataComponentUsage();
    //@@end
}
```

//@@begin javadoc:getInformation()
/**
 * Implementation of IInfo Component Interface.
 * @return Short information about what this component does.
 */

public String getInformation() {
    //@@begin getInformation()
    return "Visualizes a list of all employees.";
    //@@end
}

The following code example shows how the method of the IInfo interface can be called on a IViewer Component used. The code is located in the onActionAddViewer() method of the View DynamicView (Component Dynamic). The IExternalIInfo class is the Java interface representing the Interface Controller of the Component Interface IInfo. More generally the Java type representing the Interface Controller of a Component Interface is called IExternal<Name of the Component Interface>. The Java type representation of the Interface Controller of a Local Component Interface is called IExternal<Name of the Component>Interface.

//Call the component controller to create and initialize a new new
//viewer instance.
IWDComponentUsage viewerUsage =
    wdThis.wdGetDynamicController().createViewerUsage();

//If the created viewer component also implements the IInfo component
//interface (That's the case because the IViewer interface extends the
//IInfo interface) do a cast and call the method getInformation() on
//the IInfo interface.
if (viewerUsage.getInterfaceController().
    wdIsInstanceOf(IExternalIInfo.class)){
    info = ((IExternalIInfo)viewerUsage.getInterfaceController().
            wdCastTo(IExternalIInfo.class)).getInformation();
}

The view uses the information from the method call to provide a tooltip on the UI.
View Containers in Interface Views

For a long time, there has been the option of defining view containers at the view level using an IWDViewContainerUIElement. These view containers are placeholders for external UIs (other Views or Interface Views) that can be filled in the Navigation Modeler of a Window.

As of SAP NetWeaver 7.1* it also possible to define such placeholders in Interface Views which can then be used in the same way as view containers in local Views.

This feature is used in the tutorial application by the layout Components RowLayout and ColLayout, both implementers of the Component Interface ILayout.

In the Interface View of this Component Interface two view containers Content1 and Content2 are defined.

Now these view containers must be “implemented”. First, we have to define the view containers in the Window that implements the Interface View LayoutIfView and map these Window view containers to the corresponding view containers of the Interface View.

This is done in the Window Modeler (View Containers tab) as shown in the following screenshot for the ColLayoutWdw.

Then we define some view containers at the View level using IWDViewContainerUIElements. The following figure shows the View Modeler of the ColLayoutView (ColLayout Component) which arranges the view containers in columns surrounded by some decorative UI elements.
In the Navigation Modeler of the ColLayoutWdw Window which implements the Interface View of the Component Interface ILayout the view containers of the ColLayoutView are not filled with other UIs but selected as implementers of the defined view containers of the Window.

In the same way, the RowLayout Component is created.

The following figure shows how the Interface View of the Component Interface ILayout is embedded in the TutorialWdw Window (Tutorial Component) and that the view containers are filled with the UI of two predefined IViewer Components.
Another example of how this feature is used can be found in the Web Dynpro Java Demo Kit framework. The ILShape Component Interface defines two view containers on its Interface View for the displayed content and sidepanel.

Referencing Mode

As described above ComponentUsages allow you to reference an instance of a Component and to control its lifecycle. In some cases it is useful to share one and the same instance of a Component and not have several instances of it.

This can be achieved by setting a ComponentUsage in Referencing Mode to another ComponentUsage.

An example in the tutorial application is the Component EmplData. All viewers share the same instance of this data provider.

The Tutorial Component is the owner of the EmplData Component and controls its lifecycle. In the wdDoInit() method of the Tutorial Component the EmplData component is initialized.

```java
//Initialize the employee data provider component
wdThis.wdGetEmployeeDataComponentUsage().createComponent( Constants.EMPLDATA_COMPONENT_NAME,
   Constants.DC_NAME);
```

All Components that are interested in accessing the employee data (for example the IViewer implementers) also define their own ComponentUsage for this purpose. To enter the Referencing Mode to the source ComponentUsage (the ComponentUsage of the Tutorial Component that “contains” the EmplData Component instance) these Components would need a reference to the source ComponentUsage.

The easiest way to achieve this would be that the Tutorial Component exposes the ComponentUsage for example via the Context.

However, the recommendation is to do it the other way round. In the IViewer Component Interface the getEmployeeDataUsage() method is defined: this allows access to the ComponentUsage to the EmplData Component of a viewer Component.
You can then enter the Referencing Mode on the outside as done in the `wdDoInit()` method of the Tutorial Component:

```java
//Initialize viewer 1 component and set the reference of the viewer to
//the employee data provider component.
wdThis.wdGetViewer1ComponentUsage().createComponent(
    Constants.DEFAULT_VIEWER_1_COMPONENT_NAME,
    Constants.DC_NAME);
viewerDataUsage = wdThis.wdGetViewer1ComponentUsage().
    getController(IExternalIViewer.class).getEmployeeDataUsage();
viewerDataUsage.enterReferencingMode(
    wdThis.wdGetEmployeeDataComponentUsage());
```

This pattern has the advantage that the owner of the source `ComponentUsage` does not lose the full control of the lifecycle of the `Component` instance used. It is possible to call the methods `createComponent()` or `deleteComponent()` only on the `ComponentUsage` that “contains” the `Component` instance. `ComponentUsages` that are in Referencing Mode only can leave the Referencing Mode again (`leaveReferencingMode()` method).

This pattern is also used in the Web Dynpro Java Demo Kit framework to share the configuration.

It is also possible to enter the Referencing Mode to a `ComponentUsage` that is also in Referencing Mode to a third `ComponentUsage`. The root of such a Referencing Mode tree must be a `ComponentUsage` that is not in Referencing Mode (that can “contain” a `Component` instance).

### Component Query Service

Web Dynpro provides an API – called the `IWDCComponentQuery` – which allows searching for Components that implement a certain Component Interface.

`IWDCComponentQuery` provides several `include*()` and `exclude*()` methods to define the search space – a set of Deployable Objects (Web Dynpro Development Components).

The result of the search is a collection of `WDDeployableObjectParts` of type `COMPONENT`.

When you enter a query for the first time after the startup of the J2EE server for `Components` implementing a certain `Component Interface` it may take a few seconds until the result is processed depending on the size of the configured search space. The result is then cached for later reuse. We recommend to narrowing down the search space as far as possible using the `include*()` and `exclude*()` methods.

The tutorial application uses this API to find all available implementers of the `IViewer` Component Interface. The search space includes all Deployable Objects in this case.

The code is located in the `wdDoInit()` method of the Component `Dynamic`.

```java
//Initialize the deployable object part which represents the IViewer
//component interface and store it locally for later reuse.
try{
```
viewerInterface = WDDeployableObject.getDeployableObjectPart(
    Constants.DC_NAME,
    Constants.IVIEWER_INTERFACE_NAME,
    WDDeployableObjectPartType.COMPONENT_INTERFACE);
}catch(WDDeploymentException e){
    throw new RuntimeException(e);
}

//Create a component query and ask for all components implementing
//the IViewer component interface.
IWDDComponentQuery compQuery =
    WDQueryService.getInstance().createComponentQuery();
compQuery.includeAll(); //Incl. all depl. objects into the search space
Collection<WDDeployableObjectPart> viewerComponents =
    compQuery.findComponentsImplementingInterface(
        viewerInterface.getDeployableObjectName(),
        viewerInterface.getName());
...

WDDeployableObjectParts contains the technical details of a Component such as the name or the
Deployable Object that contains the Component. In the tutorial application, the user should be able to select
the available IViewer implementations. The name of a Component is normally not very user-friendly.
Therefore, each Component that implements the IViewer Component Interface provides a property sheet
containing the user friendly name.

The property sheet is accessed via the WDConfiguration API.

//Iterate the result and fill the context.
for(WDDeployableObjectPart viewerComponent : viewerComponents){
    IAvailableViewersElement el =
        wdContext.createAndAddAvailableViewersElement();
    String displayName = null;
    try{
        //Check whether a property sheet "viewerconfig.properties" with
        //an property "displayname" is available for the found component.
        //If yes, the value of this property is used as display name for
// the component on the UI. Otherwise the component name is used.
IWDConfiguration config = WDConfiguration.
    getConfigurationByName(viewerComponent, "viewerconfig");
displayName = config.getStringEntry("displayname");
}catch(WDConfigurationNotFoundException e){
displayName = null;
}catch(WDInvalidConfigParameterException e){
displayName = null;
}
el.setDisplayName(
    displayName == null ?
    viewerComponent.getShortName() :
    displayName);
el.setDeployableObjectPart(viewerComponent);
}

With the IWDCOMPONENTQUERY API, it is easy to implement a plug-in mechanism to enrich an application with new functionality without any code changes. For example, it is possible to implement new IViewer Components for our tutorial application in a separate Development Component. These new Components can be used automatically in the tutorial application after deployment.

Simply searching for all available Components implementing a certain Component Interface and using them in your application has the risk that “unauthorized” Components (such as test components, components that are not yet reviewed / shipped, …) can also be used immediately when they are deployed! Therefore, you should make the search space configurable (for example using .properties files). This configuration should contain the names of the “trusted” Development Components.

Dynamic Component Embedding

As described in the previous section it is possible to build a plug-in mechanism to easily consume new content in your application. Let us assume that you want to have a tab strip in your UI that should contain a tab for each available plug-in. The problem is that you do not know how many will be finally available when you build your application. In this case you have to dynamically initialize the Components found and adapt the UI. How this can be done is described here.

The code can be found in the createViewerUsage() method of Component Dynamic and in the onActionAddViewer() method of the DynamicView View (Component Dynamic). The UI is adapted in the wdDoModifyView() method of the DynamicView View.

```java
public IWDCOMPONENTUSAGE createViewerUsage() { //@@begin createViewerUsage()
    // Get a new locally unique id
    int id = counter++;

    WDDeployableObjectPart viewer =
        wdContext.currentAvailableViewersElement().
            getDeployableObjectPart();

    // Create a new component usage to the IViewer component interface and
    // initialize the component.
    IWDCOMPONENTUSAGE viewerUsage = wdComponentAPI.createComponentUsage(
        "EmbeddedComponent"+id, viewerInterface);
    viewerUsage.createComponent(
        viewer.getName(), viewer.getDeployableObjectName());

    // Set the reference of the viewer to the employee data provider
    // component.
```
IWDComponentUsage viewerDataUsage =
    viewerUsage.getController(IExternalIViewer.class).
    getEmployeeDataUsage();
viewerDataUsage.enterReferencingMode(
    wdThis.wdGetEmployeeDataComponentUsage());

    return viewerUsage;
//@@end

public void onActionAddViewer(IWDCustomEvent wdEvent) {
    //@@begin onActionAddViewer(ServerEvent)
    //Skip processing when no viewer is selected.
    IAvailableViewersElement viewerElem =
        wdContext.currentAvailableViewersElement();
    if(viewerElem == null){
        return;
    }

    //Call the component controller to create and initialize a new
    //viewer instance.
    IWDComponentUsage viewerUsage =
        wdThis.wdGetDynamicController().createViewerUsage();

    //If the created viewer component also implements the IInfo component
    //interface (That's the case because the IViewer interface extends the
    //IInfo interface) do a cast and call the method getInformation() on
    //_//the IInfo interface.
    //Store the result locally for later use in the Modify-View phase.
    if(viewerUsage.getInterfaceController().
        wdIsInstanceOf(IExternalIInfo.class)){
        info = ((IExternalIInfo)viewerUsage.getInterfaceController().
            wdCastTo(IExternalIInfo.class)).getInformation();
    }

    //Find the meta information of the interface view of the IViewer
    //component interface and of its inbound plug.
    IWDAbstractViewInfo interfaceViewInfo =
        viewerUsage.getComponentUsageInfo().
        getUsedComponent().findInInterfaceViews("ViewerIfView");
    IWDInboundPlugInfo inboundPlug =
        interfaceViewInfo.findInInboundPlugs("In");

    //Find the meta information of the usage of this view on the window.
    IWDWindowInfo windowInfo = wdComponentAPI.getComponentInfo().
        findInWindows("DynamicWdw");
    IWDViewUsageInfo sourceViewUsage =
        windowInfo.getViewUsageByID("DynamicViewUsage");

    //Create a new view container for the current view and assign the new
    //component and its interface view to this view container.
    IWDViewInfo viewInfo = (IWDViewInfo) wdControllerAPI.getViewInfo();
    IWDViewContainerInfo viewContainer = viewInfo.createViewContainer();
    IWDViewContainerAssignmentInfo containerAssignment =
        sourceViewUsage.createViewContainerAssignment();
    containerAssignment.setViewContainer(viewContainer);
IWDDViewUsageInfo targetViewUsage =
    containerAssignment.createEmbeddedViewUsage();
targetViewUsage.setComponentUsage(viewerUsage.getComponentUsageInfo());
targetViewUsage.setView(interfaceViewInfo);

    //Create a new outbound plug for the current view, link it with the
    //inbound plug of the interface view and trigger the navigation to make
    //the interface view visible on the UI.
    IWDDOutboundPlugInfo outboundPlug = viewInfo.createOutboundPlug();
    sourceViewUsage.createNavigationTarget(outboundPlug.getName(),
        targetViewUsage, inboundPlug.getName());
    wdControllerAPI.firePlug(outboundPlug, null);

    //Store the name of the new view container locally for later use in the
    //Modify-View phase.
    nameOfNewViewContainer = viewContainer.getName();
    //@@end

public void wdDoModifyView(IWDView view, boolean firstTime) {
    //@@begin wdDoModifyView
    //Adapt the UI if a new Viewer was added before.
    if(nameOfNewViewContainer != null){
        //Create a new ViewContainerUIElement and assign the new view
        //container.
        IWDDViewContainerUIElement viewContainerElement =
            view.createElement(IWDViewContainerUIElement.class);
        viewContainerElement.setViewContainer(nameOfNewViewContainer);
        //Wrap the ViewContainerUIElement with a Group.
        IWDCaption header = view.createElement(IWDCaption.class);
        header.setText(wdContext.currentAvailableViewersElement().getDisplayName());
        IWDDGroup group = view.createElement(IWDGroup.class);
        group.setHeader(header);
        group.setWidth("100");
        group.setHeight("100");
        if(info != null){
            group.setToolTipText(info);
        }
        group.addChild(viewContainerElement);
    }
    else{
        nameOfNewViewContainer = null;
        info = null;
        //@@end
    }

    nameOfNewViewContainer = null;
    info = null;
    //@@end
}
When the user chooses Add on the UI the corresponding action is triggered and the action event handler onActionAddViewer() is called.

First the createViewerUsage() method of Component Dynamic is invoked to dynamically create a new Component Usage that points to the IViewer Component Interface and initialize the new Component used.

Then, the Web Dynpro runtime metadata for the view is extended:

1. Search for the necessary information about the Interface View of the IViewer Component Interface and about the current View Usage of the view in the Window.

To access the View Usage via the APIs shown from the Window (via name "DynamicViewUsage") the name has to be defined in the Navigation Modeler of the Window:

2. Create a new view container for the current view and assign the new component and its interface view to this view container.

3. Create a new Outbound Plug on the View and link it to the Inbound Plug of the Interface View of the embedded Component.

4. Trigger the Outbound Plug to make the embedded Component visible on the UI.

5. Remember the name of the view container created for adapting the UI in the Modify View Phase.

Finally in the wdDoModifyView() method a new IWDViewContainerUIElement is created, the view container created is attached to this UI element and it is added together with some decorative elements as a child to a existing IWDScrollContainer.

Another example of how the described code is used can be found in the Web Dynpro Java Demo Kit framework (Component CompAdaptComp in Development Component tc/wd/uidemokit/adpt).
Componentization

The “Componentization” section deals with the question of how you should split up your Web Dynpro application code in the best way into several Web Dynpro Components and also into several Development Components.

Unfortunately the answer to this question is “It depends on your project”. This section provides you a short overview of some basic principles for good componentization. These principles can only be hints. As already mentioned, it depends on your concrete project as to how strictly you should follow these principles (How big is the application? How much reuse should be possible? How easy should it be to adapt/extend the application once it is finished?).

Components are the reusable entities in Web Dynpro. Before you start your project, you should think about which Components you want to create. It makes sense to separate code and UI parts in Components that can be reused in several locations in your application as well as in other applications (for example a UI which visualizes address data or a Component that accesses a back-end system and provides the data). In addition to the aspect of reusability, you should also think of the size of a Component. A Component that contains many Views, Custom Controllers, Windows and so on is hard to overlook. But the same is true for a project that contains many small Components.

Another principle is to use faceless Components as data providers to encapsulate back-end accesses, Web Services and so on. These Components can then be shared and referenced from different other Components in your project to access the data (such as the EmplData Component in the tutorial application).

Component Interfaces allows separating between the interface definitions and the concrete Components that implement these interfaces. This allows you to simply exchange Component implementations (for example using another data provider Component in the tutorial application) or add new implementations (for example deploying new viewers in the tutorial application) without the need of refactoring your complete project.

More articles and tutorials regarding this topic can be found on SDN, for example. For more information, see the links in the Further Information section.
Further Information

- **SAP NetWeaver CE 7.1 EHP1 Help (Web Dynpro Components)**
  [http://help.sap.com/saphelp_nwce711/helpdata/en/42/284ec114c840b3a2eb0c6bc30ab47d/frameset.htm](http://help.sap.com/saphelp_nwce711/helpdata/en/42/284ec114c840b3a2eb0c6bc30ab47d/frameset.htm)

- **SAP Professional Journal, Leverage component-based architecture in Web Dynpro Java business applications (article series), by Bertram Ganz, Senior Product Specialist, SAP AG, and Richard Tucker, Principal Web Development Architect, Atos Origin UK (NOT FREE OF CHARGE):**

- **Available SDN Content dealing with Components and Componentization:**
  - Web Dynpro Java Controller and Interface Concept: [http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/f0ba2c45-0518-2a10-73be-9b785e10ae1](http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/f0ba2c45-0518-2a10-73be-9b785e10ae1)
  - Componentization Exercise in Web Dynpro Java: [http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/30d56578-f7e5-2a10-5ab5-8afc55d2e0e8](http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/30d56578-f7e5-2a10-5ab5-8afc55d2e0e8)

- **Web Dynpro Forum on SDN**

- **Web Dynpro Java Components in NW CE 7.1:**
  Document *NewWDJavaComponenModel_NWCE71*
# Legend

## Text Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Symbol" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="image2" alt="Symbol" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="image3" alt="Symbol" /></td>
<td>Warning</td>
</tr>
<tr>
<td><img src="image4" alt="Symbol" /></td>
<td>See also</td>
</tr>
</tbody>
</table>
Copyright

© Copyright 2010 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, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, zVME, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER5+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated 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.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, 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 other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

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.