Migrating Web Dynpro Application Projects from SAP NetWeaver 7.0 to Higher Version

Applies to:

Migration of Web Dynpro for Java applications running on top of the SAP NetWeaver 7.0 release to the SAP NetWeaver 7.1x and 7.20 releases. For more information, visit the Web Dynpro Java homepage.

Summary

The goal of this document is give an comprehensive overview how to migrate Web Dynpro for Java applications running on top of the SAP NetWeaver 7.0 release to the SAP NetWeaver 7.1x and 7.20 releases.

Author: SAP Web Dynpro for Java Runtime Team

Company: SAP

Created on: June 2010

Table of Contents

Migrating Web Dynpro Application Projects from SAP NetWeaver 7.0 to Higher SAP NetWeaver Version

The general migration guide for migrating application projects from SAP NetWeaver 7.0 to higher SAP NetWeaver Version has not been released yet. This document will be updated, once the general guide is available.

Outline of steps not covered here:

- Migrating Tracks Content from SAP NetWeaver 7.0 to Higher Version
- Migration from JDK 1.4.2 to JDK 5/6
- Run the migration wizard to automatically adapt the old projects' nature (for example, projects' metadata, artifacts, and structure) to the new one.

Migrating Web Dynpro Applications

Use

In order to use the new functionality found in SAP NetWeaver CE 7.1 and higher you need to migrate your SAP NetWeaver 7.0 Web Dynpro applications to the new architecture.

Complete migration of Web Dynpro Java application written for SAP NetWeaver 7.0 is not mandatory. In SAP NetWeaver CE 7.1 and higher there is source code compatibility of Web Dynpro Java applications but not binary compatibility. This means that the old source code must be imported, re-compiled, and redeployed to the new system. Applications created in or migrated to NetWeaver CE 7.1 are binary compatible to NetWeaver CE 7.2. You can directly deploy an application built for CE 7.1 on a CE 7.2 runtime. However, this does not work in the opposite direction. You cannot deploy an application built for CE 7.2 on a CE 7.1 runtime.

Prerequisites

You have imported the Web Dynpro development components into the SAP NetWeaver Developer Studio.

Procedure

1. Migrate Web Dynpro Development Components (DCs).
   For more information, see Migrating Web Dynpro Development Components
2. Deploy Web Dynpro DCs.
   For more information, see Deploying Applications.
3. Run the Web Dynpro application.
   For more information, see Running Web Dynpro Applications.

Migrating Web Dynpro Development Components

Use

Use this procedure to reuse the DCs available in the SAP NetWeaver 2004 and SAP NetWeaver 7.0 environment in the latest SAP NetWeaver CE 7.1 or higher.

The migration task includes a few subtasks that are mandatory for building the Web Dynpro DCs. The remaining optional subtasks need to be performed if you want to make use of the new functionality provided by the higher SAP NetWeaver CE 7.1 or higher.

You can find details related to migration considerations and other noteworthy information from the following topics:

- Recommendation Whether to Migrate a Component
- Recommendation Whether to Migrate a Component Interface Definition (CID)

- Noteworthy Information on Migration and Changes between the SAP NetWeaver 7.0 and subsequent releases

Procedure

1. Migrating DC dependencies

For more information, see Running the Application Projects Migration Wizard [Link not available].

2. Migrating Internal APIs to Public APIs

We recommend you **do not** use internal APIs as they are subject to change at any time and also can undergo semantic changes without warning. For more information, see SAP Note 928485.

1. Choose the project that needs API migration.
2. Using the secondary mouse button, choose *Repair → Internal Web Dynpro APIs usage*.

3. Optional: Migrating Component Interface Definitions

An existing Web Dynpro Component Interface Definition (CID) should only be migrated to the new component model in specific cases.

For more information, see Recommendation Whether to Migrate a Component Interface Definition (CID).

1. Select the CID to be migrated in the SAP NetWeaver Developer Studio.
2. Using the secondary mouse button, choose *Migrate*.

4. Optional: Migrating Components

An existing Web Dynpro Component should only be migrated to the new component model in specific cases.

For more information, see Recommendation Whether to Migrate a Component

1. The WD components, which have not been migrated are displayed in grey. Select the grey-colored Web Dynpro component to be migrated in the Developer Studio.
2. Using the secondary mouse button, choose *Migrate*.

   The tool opens a Cheat Sheet which guides you with the list of instructions to migrate the components.

   For more information, see Cheat Sheet

5. Optional: Migrating Model

1. Migrating Adaptive RFC Model to Adaptive RFC 2 Model
2. Migrating to the Adaptive Web Service Model Type

6. Optional: Migrating Fusion

1. Start the console by *<instance-dir>\SYS\global\com.sap.dtr.console\run.bat*.
2. Connect to the Design Time Repository (DTR) containing the source track with the unmodified Web Dynpro project using the connect command (help connect).
3. Connect to the DTR containing the target track which should contain the migrated Web Dynpro project.
4. Call *copyAsFusion -targetSession <targetDtrSession> -r .*\.wdmodel,.*\.wdcomponent /ws/<sourceTrackname>/<SCName>/dev/inactive/DCs /ws/<targetTrackname>/<SCName>/dev/inactive/DCs <activityname>*.

You need to rebuild your project after you have completed the above migration steps.

Cheat Sheet

Preparing your Web Dynpro Components for Migration

Move Views

The new programming model for Web Dynpro components requires that all views of the component are physically stored within the same package as the component itself. The views for which the packages have to change are listed in this step of migration.

Move Custom Controller

The new programming model for Web Dynpro components requires that all custom controllers of the component are physically stored within the same package as the component. The custom controllers for which the packages have to change are listed in this step of migration.

Move Windows

The new programming model for Web Dynpro components requires that all windows of the component are physically stored within the same package as the component itself. The windows for which the packages have to change are listed in this step of migration.

Conflicting Windows Name

The new programming model for Web Dynpro components requires that all windows of the component have a window controller. These controllers will be created during migration. On the other hand there must not be two controllers with the same name within one Web Dynpro component. The step lists windows for which this constraint is violated. The window or the object with the conflicting controller has to be renamed.

Keep in mind that renaming an object that is visible from the outside might break the build of other components that use the component being migrated.

Component Package Check

The new programming model for Web Dynpro components requires that you reserve a package solely for a component and its sub-objects. The term “sub-objects” refers to windows, views and the local component interface controller. This step lists the objects found within the package of the component that might conflict with this requirement. Either the component has to be moved into another package or the conflicting objects need to be refactored.

5. Moving a component might break its usage by other components.
6. This step lists two alternatives. Either the component can be moved or other objects can be moved.
   - If you choose to move the component, then other objects need not be moved. Simply mark the movement of those objects as "done" by clicking on the status icon.
   - If you choose to move the other objects then they must be moved manually. Mark the status as "done" by clicking on the status icon. After all the objects are moved, simply click on the step of movement of the component
7. In step 2.b: If all the objects are not moved and the steps are marked as done, then clicking on the icon for movement of the component will trigger this action. If all the objects are moved and the steps are marked as done, clicking on icon for movement of the component will simply mark it as done.

Check Native Interface Inheritance

During the migration of the component it is necessary to let the component local interface extend all implemented interfaces and to enable 'native inheritance' for the component local interface. This needs to be done in order to keep the programming interface compatible with the version before the migration. On the other hand 'native inheritance' must not be enabled if one of the inherited interfaces has a version that is too old (in other words: was created with NetWeaver Developer Studio 04 or 04s).

The standalone component interfaces listed below bear such a conflict. These conflicts can be solved in one of two ways: either migrate the inherited component interface to the newer version (using the context menu function 'Migrate' for the interface) or create a new standalone interface, let this interface extend the problematic one and implement the new interface.

Migrating the component interface will also slightly change the runtime behavior of the interface, which might have consequences for any components that use the one containing the migrated interface. It is no longer possible to cast from Java interface corresponding to a Web Dynpro component interface to another one, as there are separate runtime representations for each implemented Web Dynpro component interface.

Implement Interface Controller

Within the new programming model for Web Dynpro components, the component controller is required. This component controller is used to implement the following:

- An interface controller belonging to the component's local interface
- An interface controller of any used standalone component interfaces

In order to do so, the declaration of all objects within these interface controllers must be replicated into the component controller. This cheat sheet thus helps you to copy all the relevant objects into the component controller. After that, only the relevant coding contained within the component's local interface controller needs to be moved to the component controller. Moving the coding may require that it be adapted. Use 'Check implementation relation' to find out which objects defined in the interface controller are not implemented within the component controller and if there are any conflicts. You can then use 'Implement interface controller' to automatically copy all relevant objects from the local interface controller to the component controller in order to fulfill the necessary implementation relation.

Hint: If the context of the interface controller contains nodes with context mappings to the component controller this will create implementation errors such as 'The type of the attribute in the interface definition is missing'. To solve this, simply delete the mappings from the context nodes in the interface controller to the component controller.

Transfer User Coding Areas

The step lists the user coding sections of the local component interface controller which may contain Java coding that needs to be transferred to the component controller. These user coding areas have to be copied manually to the appropriate sections of the component controller.

Replace/Delete References

The new programming model for Web Dynpro components requires that the implementation of all interface coding and the storage of data is contained within the component controller instead of the interface controller. This means that all references to methods and data previously held in the interface controller must be replaced with references to the corresponding methods and data that have now been transferred to the component controller.

After performing this step, the component will not run any more. If you redefined some of the context elements during the previous step, the list might be out-of-date. If this is the case, restart the cheat sheet activity.

Delete Implementation Coding from Interface Controller

In the new programming model for Web Dynpro components there is no possibility to implement any coding in the component's local interface controller. The coding that was stored there in the old programming model must first be transferred to the component controller (as described in the previous steps). Now the redundant coding in the interface controller can be deleted. Click the item in the cheat sheet to perform this automatically. Make sure you have transferred all necessary coding to the component controller before proceeding!

After performing this step the component will not run any more.

Delete Implementation Metamodel Objects

Once the coding stored in the component's local interface controller has been deleted all that may remain are some metamodel objects such as controller usages, supply functions and calculated attribute provider methods. These objects will now be deleted by performing this step. Make sure that all these objects have been transferred before to the component controller before proceeding!

After performing this step the component will not run any more

Start Migration

If all the previous steps are executed successfully the component is ready for migration. You are now ready to execute the step that actually migrates the component.

Any coding from the Interface View Controller is not copied to the Window Controller. This coding must be manually transferred.

Recommendation Whether to Migrate a Component Interface Definition (CID)

SAP recommends not migrating existing Web Dynpro Component Interface definitions by default. In most existing Web Dynpro application scenarios there is no necessity for component interface definition migration.

Migration is mandatory, if

- The Component Interface Definition is to be evolved using new features (component interface inheritance and view containers in interface views). However, keep in mind that such changes to the component interface definition require changes on all components implementing that interface.

Migration is optional, if

- Web Dynpro developers are not familiar with the old Web Dynpro component model.

Recommendation Whether to Migrate a Component

SAP recommends not migrating existing Web Dynpro component by default.

SAP recommends not migrating existing Web Dynpro components to the new Web Dynpro Component model of NetWeaver CE by default. This means that in most cases the Web Dynpro Runtime and Designtime environments support all existing Web Dynpro components and it is possible to develop cross-component scenarios containing both Web Dynpro component types.

In other words, you can define usage dependencies between old and new Web Dynpro components in order to embed component interface views, define context mapping relations, subscribe event handlers or invoke the interface controller's IExternal-API.

This recommendation is motivated by the fact that in most cases the component migration process cannot be fully automated by the *Component Migration Tool* and that therefore, some potentially time consuming effort will be required to complete the migration. Additionally existing Web Dynpro components will benefit from long term support so that component migration step is not mandatory.

Note that the migration to the new concept of universal context elements is independent of the migration of a Web Dynpro component. So neither is it necessary to migrate to the new component model to make use of the universal context elements, nor is the switch to the new universal context elements done as a part of the component migration.

There are only a few specific use cases in which SAP recommends the migration of existing Web Dynpro components

Migration Use Case	Description	Migration Decision
An existing Web Dynpro component must implement a **new standalone component interface definition** utilizing new functions.	You need to assume a multi-component Web Dynpro application architecture which makes use of Web Dynpro component interface definitions. You have to further assume that this component interface definition is based on the new Web Dynpro component model. In case a new component interface definition applies a new feature of the new Web Dynpro component model (like view containers in component interface views or component interface inheritance) all implementing Web Dynpro components must also be based on the new component model. Without migrating existing Web Dynpro components to the new component model they cannot be used as implementing components inside an enhanced Web Dynpro component architecture.	Mandatory
An existing Web Dynpro component must implement a **new standalone component interface definition** not utilizing new functions.	Existing Web Dynpro components are compatible with the new standalone component interface definition model as long as the component interface definition does not utilize a new feature. This means existing Web Dynpro components (based on the old component model) can still implement or are compatible with a new component interface definition (based on the new component) model, as long as no new functions are applied in this component interface definition.	Not necessary
An existing Web Dynpro component must utilize **component interface inheritance,** which is only supported in the new component model	Component interface inheritance can only be applied in the new component model. To embed an existing Web Dynpro component in a Web Dynpro component architecture utilizing component interface inheritance together with standalone component interface definitions, the existing component must be migrated.	Mandatory
An existing Web Dynpro component must support the generic **secondary help** service	To utilize the new *secondary help service* in Web Dynpro application running in the NW portal you must define help links at the window controller level inside a Web Dynpro component. Help links cannot be defined in NW04/7.0 component model.	Mandatory
Web Dynpro component developers are not familiar with the old Web Dynpro component model	Depending on the technical knowledge of your Web Dynpro application developers it might be advisable to migrate existing components to the new component model. In case a Web Dynpro application developer is not familiar with the WD component model of NW 04 and NW 04s the total cost of learning the old architecture might outweigh the total cost of component migration. The component migration must be done by a Web Dynpro	Optional

	developer who is familiar with both the old and new Web Dynpro component models	
You want to simplify your **cross-component navigation** by applying the new window navigation plug concept	The new Web Dynpro component model allows you to define inbound and outbound plugs at the window level. This greatly simplifies navigation across component borders, because a navigation event triggered at the window level does not require server-side eventing combined with a navigation dispatcher view as was formerly the case in NW04 and 7.0. Since plugs can now be defined inside windows and fired by the new window controllers, the implementation of cross-component navigation logic is much simpler. Nevertheless, this simplification does not make component migration mandatory.	Optional
Utilizing **View Containers in Component Interface Views**	*View Containers in Component Interface Views* are an additional function in the new Web Dynpro component model. They are needed to develop so-called *Web Dynpro layout components* in NW CE. *A Web Dynpro layout component* allows you to embed and position other visual Web Dynpro components without needing to manage the related component instance lifecycles. In NW 04 and NW 04s it was not possible to develop such layout components because it was not possible to decouple the lifecycle management of component instance from a component interface view embedded in separate components. Instead, the layouting of component interface views had to be defined within the root component because this is where the instance lifecycles of its embedded visual components were managed. Based on the above facts, there is no practical use case for migrating an existing Web Dynpro component to the new component model simply to extend it to a Web Dynpro layout component. It makes no sense to migrate an existing root component to a layout component. Instead, such a Web Dynpro layout component should be newly created from scratch. This avoids the need to perform component migration. Migration is only mandatory, in cases where an existing Web Dynpro component must implement a new component interface definition utilizing new functions.	Not necessary (but see use case description)
You want to apply the new **default plug** property for Web Dynpro view usages	In NW CE it is possible to define a *default plug* for all views embedded in a Web Dynpro Window either in a window directly, in a view area or in a ViewContainerUIElement inside a	Not necessary

	layout view. The purpose of the default plug is to let the Web Dynpro Runtime start the related inbound plug event handler **without** first needing to traverse a navigation link. As this new feature is also applicable for all view layouts inside existing Web Dynpro components, a component migration is not required. ***Example*** A view controller has the lifecycle property of framework_controlled. When the view layout is first displayed in a view assembly, the Web Dynpro Runtime creates the view controller instance and initializes it by invoking its wdDoInit() hook method. When the view layout disappears from the view assembly (as a result of navigation) the lifecycle property value causes the view controller instance to persist. Consequently the wdDoInit() hook method will not be invoked again when the view layout re-appears in a later view assembly[1]. To implement application logic for the re-appearance of a view layout which is not based on navigation link pointing to this view you can apply the default plug concept. By defining a default plug you ensure that the associated inbound plug event handler is invoked by the Web Dynpro Runtime whenever the view becomes a member of a new view assembly. You need to be aware that a view layout can also become visible in a view assembly without a navigation link directly pointing to it. This is the case if a view layout is embedded in a ViewContainerUIElement or it has been embedded in a component interface view.	

[1] The wdDoInit() hook method of a Web Dynpro controller is only ever called once, at the start of the controller's lifecycle.

Noteworthy Information on Migration and Changes between the SAP NetWeaver 7.0 and subsequent releases

There are several changes which might have an impact when running applications on the new releases:

- There is a new URL scheme for Web Dynpro applications in place (…/webdynpro/resources/<dc-vendor>/<dc-name>/<app-name> instead of …/webdynpro/dispatcher/<dc-vendor>/<dc-name>/<app-name>). However, the old URL scheme still works for compatibility reasons. Furthermore, resource paths have changed in order to provide better support of the Web module.
    - Applications which use the WDURLGenerator API as required will continue to get the correct URLs.
    - Applications which create URLs using some custom written algorithm will need to be checked for compatibility as they might experience trouble in certain scenarios.
- At the end of a controller's lifecycle context elements are now cleaned up earlier. This might lead to runtime exceptions, if elements of the context are accessed after they have become invalid. Such accesses have been an error already in 7.0, but could have gone unnoticed.
- Due to security reasons, the input data is checked more thoroughly by the server starting with 7.11. This might lead to runtime exceptions if an application relies on the more relaxed behavior of the framework in 7.0.
- In 7.20 additional security improvements have been applied which might break existing test tools, e.g. for load tests.
- Starting with 7.1, the Unified Rendering (UR) library called "Lightspeed" is used as default UI rendering library. UR Classic is no longer available from 7.2 onwards. This has especially an impact on test tools relying on the used HTML.
- Using Java native types instead of the DDIC wrapper types for a view element's primary property makes the corresponding view element read-only. Using the DDIC wrapper was mandatory in 7.0 but had not been enforced.
- Input fields with a numeric type show a calculator for value help starting in 7.11.
- Starting from 7.10, the lead selection change behavior in tables has slightly changed for components using the old component model. Previously the lead selection was directly changed when selecting a table row. Now, the lead selection is only changed after the next server roundtrip. This implies that in a master-detail scenario, the detail form/table still shows the data for the old lead selection, while in the master table the data for the new row is edited. For components using the new component model, there are even more changes to the lead selection handling.
- There is a new table column concept. It is important not to mix new and old columns within the same table. Manual migration of the columns to the new concept is optional.
- There is a new context menu concept. This might interfere with the old context menus specifically available for some view elements (i.e. only one of the two context menus is shown at a time). So replacing the old context menus by the new concept migh have advantages.
- There are several new ways for integrating Web Dynpro applications into the SAP Enterprise Portal. Please see the SAP Enterprise Portal documentation and note 1383062 (especially the attachment).

Migrating Web Dynpro components to the new component model will have the following side effects:

- It is an irreversible Process: There is no way back once the Web Dynpro component has been migrated to the new programming model, you cannot later revert to the original programming model.
- Behavioral Changes: The migration process might slightly change how the Web Dynpro component behaves at runtime. Therefore you will need to test your component's functionality extensively after the migration. This is particularly relevant for reusable components, especially if they are stored in separate Web Dynpro Development Components.
- Migrate those components that are heavily used by other components with extreme care!

- Hook Method Signature Changes: Beginning for Web Dynpro Components created in / migrated to 7.10, the hook doModifyView() is no longer a static method. Starting with Web Dynpro Components created in / migrated to 7.11 the two parameters available as local variables (wdThis and wdContext) are no longer part of the signature of the hook method.
- Hook Method Invocation Changes: For components with the new programming model the hook methods doBeforeNavigation() and doPostProcessing() will only be called when the round-trip triggers an action event handler. Prior to the migration, these hook methods were always called. This may affect the runtime behaviour of your component and cannot be checked by the migration tool.
- Context Specific Changes: Do not invoke IWDNode.bind() for context nodes that have a supply function in the new programming model, calling the bind() method for a context node with a supply function will cause a runtime exception. This may affect the runtime behaviour of your component and cannot be checked by the migration tool.
- Table UI Element Changes: The Table UI element has a slightly different runtime behaviour for the new programming model. When the user clicks on a button within a row that is not selected, the lead selection will only change if the table is in read-only mode. This may affect the runtime behaviour of your component and cannot be checked by the migration tool.
- For the Calendar UI Element, the new aggregation WeekDayPattern takes precedence over the old RecurrenceDayPattern. If aggregation WorkingTimes contains entries, they are considered for rendering the UI Element.
- For the DateNavigator the new property legendId takes precedence over the old aggregation Legend. Starting with 7.10, the property legendId has to be specified instead of using aggregation Legend.
- New Naming Convention for UI Element IDs: In the new programming model, programmatic creation of UI element IDs that start with '_' (underscore) is no longer permitted. Doing so will cause a runtime exception. This may affect the runtime behaviour of your component and cannot be checked by the migration tool.

The following table contains UI element feature deprecations. Consider them when migrating Web Dynpro Components or creating new Web Dynpro Components. When a Web Dynpro Component is created, it gets the current version of the used IDE. If a Web Dynpro Component is migrated to the new component model, its version is changed to the then current version of the used IDE.

Special care has to be taken for deprecated aggregations. Therefore, they have already been mentioned at the beginning of this section. If a cell in the feature column contains no value, the whole view element is deprecated.

Library / View Element	**Feature**	**Name**	**Deprecated Since Version**	**Reason**
Adobe				
InteractiveForm	Property	displayType	7.20	Use display type "native" instead
BusinessGraphics				
BusinessGraphics	Property	alwaysDisplayGraphic	7.10	Table rendering mode in the accessibility mode is deprecated. The application developer is responsible to choose a suitable way to present the data of the Business Graphic in an accessible way.
BusinessIntelligence				
GridCell			7.11	Don't use this UI element anymore.
GridCellEditor			7.11	Don't use this UI element anymore.
GridCellStyle			7.11	Don't use this UI element anymore.
GridCellStyle	Property	borderStyleBottom	7.11	Don't use this UI element anymore.
GridCellStyle	Property	borderStyleLead	7.11	Don't use this UI element anymore.
GridCellStyle	Property	borderStyleTop	7.11	Don't use this UI element anymore.
GridCellStyle	Property	borderStyleTrail	7.11	Don't use this UI element anymore.
GridCellTemplate			7.11	Don't use this UI element anymore.
GridCellTextStyle			7.11	Don't use this UI element anymore.
GridCellTextStyle	Property	fontStyle	7.11	Don't use this UI element anymore.
GridCellTextStyle	Property	fontWeight	7.11	Don't use this UI element anymore.
GridCellTextStyle	Property	textDecoration	7.11	Don't use this UI element anymore.
Grid			7.11	Don't use this UI element anymore.
Celendar				
RecurrenceDayPattern	Aggregation	WorkingTimes	7.10	The binding model has

				changed. Use aggregation workingTime instead.
AbstractCalendar	Aggregation	RecurrenceDayPatterns	7.10	Use aggregation WeekDayPattern instead for static recuring working times
Mobile				
RFIDReader	Event	onRead	7.10	Use onTrigger event with property triggerMode = WDTriggerMode.read instead.
Pattern				
PatternSequenceStep			7.10	PatternSequenceStep
PatternSequence			7.10	Unsupported UIElement. Use the Guided Procedure Pattern instead (with RoadMap).
RealtimeMessaging				
MessageBasedTrigger	Property	messageName	7.10	Has been renamed to <code>eventName</code>.
Standard				
AbstractMasterTableColumn	Property	design	7.20	Use cellDesign property instead.
AbstractTableColumn	Event	onAction	7.10	This action was introduced to provide the possibility to sort or select a column. Due to there is a Table.onColSelect and Table.onSort this action is deprecated. Use Table.onColSelect or/and Table.onSort instead.
AbstractTreeNodeType	Property	iconAlt	7.10	Instead of the iconAlt text use the tool tip of the TreeNodeType or TreeItemType.
DateNavigatorLegend			7.10	Use the UIElement Legend instead of the DateNavigatorLegend. Connect the Legend and the DateNavigator via legendId of the DateNavigator.
DateNavigatorLegend	Property	category	7.10	Use the TableCellDesign enumeration instead to mark a day with a color.

DateNavigatorMarking	Property	category	7.10	Use the property daySemantics instead to mark a certain day. The Legend UIElement is used to describe the semantic meaning of daySematics color code.
Popin	Property	visible	7.11	This boolean property named "visible" has been replaced by a property named "visibility" of enumeration type in order to support the same functionality as elsewhere. As long as this property is set to false or bound to a context attribute of value false, the "visibility" property has no effect and the view element remains invisible!
RoadMapStep	Property	design	7.10	Instead of the design property use the enabled and the type property of the RoadMapStep and the selected property of the RoadMap.
Tab	Property	visible	7.11	This boolean property named "visible" has been replaced by a property named "visibility" of enumeration type in order to support the same functionality as elsewhere. As long as this property is set to false or bound to a context attribute of value false, the "visibility" property has no effect and the view element remains invisible!
TableColumn	Property	design	7.10	Use cellDesign property instead.
AbstractCaption	Property	imageAlt	7.10	Instead of the imageAlt text the tool tip is displayed if the image isn't available.
AbstractInputField	Property	displayLabelAsDefault	7.11	This property is deprecated since NW 7.11, please use property inputPrompt instead.
Button	Property	size	7.10	Small UIElements are not supported anymore.

DateNavigator	Aggregation	Legend	7.10	Instead of the Legend aggregation to DateNavigatorLegend use the property legendId to connect the UIElement Legend with the DateNavigator. The DateNavigatorLegend is deprecated.
DropDownByIndex	Property	size	7.10	Small UIElements are not supported anymore.
DropDownByKey	Property	size	7.10	Small UIElements are not supported anymore.
FileDownload	Property	data	7.10	Use the new property "resource" instead.
FileUpload	Property	data	7.10	Use the property "resource" instead.
FileUpload	Property	fileName	7.10	Use the property "resource" instead.
FormattedTextView	Property	hAlign	7.10	Use the hAlign property of the containing LayoutData instead (e.g.: MatrixHeadData.hAlign, MatrixData.hAlign, RowHeadData.hAlign).
HorizontalGutter	Property	hasRule	7.10	Use attribute ruleType instead. DisplayType=none is equivalent to hasRule=false and displayType=areaRule is equivalent to hasRule=true.
Image	Property	alt	7.10	Instead of the imageAlt text the tool tip is displayed if the image isn't available.
InputField	Property	size	7.10	Small UIElements are not supported anymore.
Link	Property	size	7.10	Small UIElements are not supported anymore.
Table	Property	compatibilityMode	7.10	There will be no alternative functionality available in the future. The two values in the enummeration have the same effect for applications implemented in releses later than NW04s.
Table	Aggregation	Columns	7.10	Functionality is covered in "groupedColumns". Use "groupedColumns" aggregation instead.

Table	Aggregation	MasterColumn	7.10	Functionality is covered in "rowArrangement". Use "rowArrangement" aggregation instead.
Tree	Property	defaultItemIconAlt	7.10	Instead of the defaultItemIconAlt text use the tool tip of the TreeItemType.
Tree	Property	defaultNodeIconAlt	7.10	Instead of the defaultNodeIconAlt text use the tool tip of the TreeNodeType.
FlowLayout	Property	defaultPaddingBottom	7.10	No alternative available.
FlowLayout	Property	defaultPaddingLeft	7.10	Use cellDesign of FlowLayouData instead.
FlowLayout	Property	defaultPaddingRight	7.10	Use cellDesign of FlowLayoutData instead.
FlowLayout	Property	defaultPaddingTop	7.10	No alternative available.
FlowData	Property	paddingBottom	7.10	No alternative available.
FlowData	Property	paddingLeft	7.10	Use cellDesign instead.
FlowData	Property	paddingRight	7.10	Use cellDesign instead.
FlowData	Property	paddingTop	7.10	No alternative available.

Related Contents

For more information, visit the Web Dynpro Java homepage.

Copyright