SDN Contribution

# Best Practices for Building Client-Independent Web Dynpro UIs

## Applies to:

Web Dynpro for Java UI Development, SAP NetWeaver 2004, SAP NetWeaver 2004s

## Summary

This article discusses best practices that facilitate the development of client-independent Web Dynpro User Interfaces. These tips on *layout*, *sizing*, *scrolling,* and *coding* will help you to build flexible, easily maintainable and even faster Web Dynpro applications.

**Name:** Bertram Ganz

**Company:** SAP AG

**Created on:** 17 May 2006

## Author Bio

After his studies in mathematics, physics and computer science Bertram Ganz finished his teacher training at a German grammar school stressing technical sciences. He has been a member of the Web Dynpro Java Runtime development team (SAP NetWeaver ESI Foundation UI) since 2002. The main focus of his work is on knowledge transfer, rollout, and documentation. Bertram regularly publishes articles on Web Dynpro in the context of the SAP NetWeaver Application Server. He also runs several Web Dynpro trainings and is the co-author of the SAP Press Book, Java Programming with the SAP Web Application Server.

## Table of Contents

## Introduction

An outstanding advantage of the Web Dynpro UI technology is its client independence. This means that the Web Dynpro programming model is agnostic to the client technology like HTML (Internet Explorer, Mozilla Firefox), Smart or Rich Client (Web Dynpro for Java, Web Dynpro for Windows), or PDAs (Mobile Web Dynpro) (see figure 1).

Based on a common *Web Dynpro UI Element Library*, its *UI event model* based on *action binding* and the principle of *data binding* from UI element properties to the context as a controller storage place, Web Dynpro applications follow the "*create once and run anywhere*" principle that decisively improves the developer productivity and flexibility to run on different and even future clients without modification.

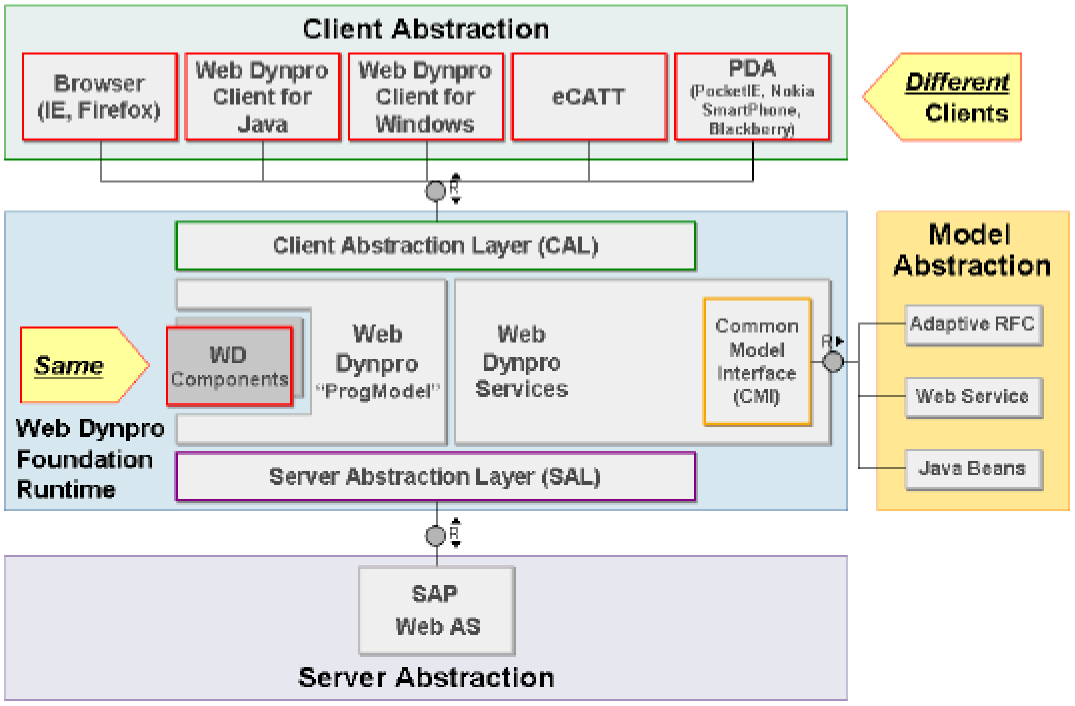

Client Abstraction
Browser (IE, Firefox)
Web Dynpro Client for Java
Web Dynpro Client for Windows
eCATT
PDA (PocketIE, Nokia SmartPhone, Blackberry)
Different Clients
Client Abstraction Layer (CAL)
Model Abstraction
Same
WD Components
Web Dynpro "ProgModel"
Web Dynpro Services
Common Model Interface (CMI)
Adaptive RFC
Web Service
Java Beans
Web Dynpro Foundation Runtime
Server Abstraction Layer (SAL)
SAP Web AS
Server Abstraction

*Figure 1 – The Web Dynpro Client Abstraction Layer*

To optimally take advantage of Web Dynpro's UI client independence as an application developer, you should follow some *Best Practices* presented in this article. Separated into the topics *General*, *Sizing*, *Laying out*, *Scrolling,* and *Coding,* these rules will sharpen your Web Dynpro skills and help you develop Web Dynpro applications that are reusable on different clients and easy to maintain.

## General Rule

### Never, Ever Misuse Web Dynpro UI Elements

Do not modify, use, or combine UI elements for a purpose they have not been designed for. This means do not try to produce a certain display result of a UI element by setting *improper* property values. An example is trying to produce an empty Group UI element in order to get a highlighted header text.

The appearance of Web Dynpro UI elements is subject to change. With the Web Dynpro UI Element technology, changes to the controls can be applied centrally, without having to edit individual screens. Misused Web Dynpro UI elements will most probably be displayed other than before in this case.

## Sizing Rules

### Consider Different Clients and Different Font Sizes

When sizing Web Dynpro UI-elements within view layouts you should always consider the fact that the absolute size of Web Dynpro UI elements could change because of localization changes, varying font sizes and even dictionary type metadata (field lengths).

Furthermore your Web Dynpro application can run on different clients that calculate box sizes like widths, heights, paddings, borders, or margins with client-specific algorithms and therefore client-specific results.

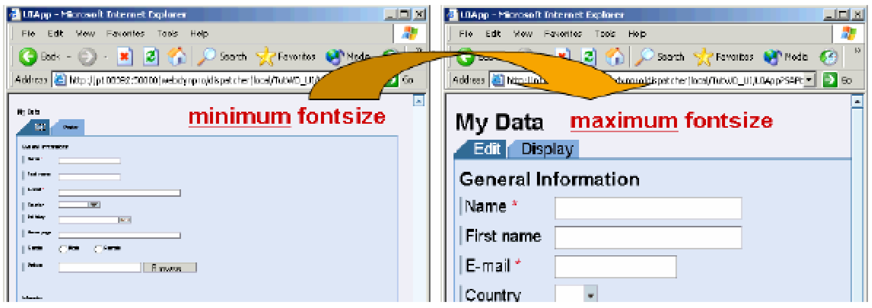


*Figure 2 – Changing the font size within a Web Dynpro browser client (CTRL – wheel scrolling)*

### Abandon Pixel Accurate Arrangement of UI Elements

To develop client-independent Web Dynpro applications you should principally abandon a *pixel accurate* arrangement of all UI elements within a Web Dynpro application window.

It is a much better strategy to utilize Web Dynpro's various layout designs and layout properties for positioning and spacing UI elements in a predefined way than it is to "hack" your own view layouts by combining *TransparentContainers* and *InvisibleElements* with absolute sizes (see next section on *Layout Rules*).

### Avoid Setting Explicit Widths and Heights

You should principally avoid specifying the UI elements' *width*- and *height*- properties at all. Better trust in the *intrinsic* sizing capabilities of your Web Dynpro Client than in your own explicit size definitions. *Intrinsic sizing* means that Web Dynpro (the Web Dynpro UI renderer and the Web Dynpro Client) adequately specifies these sizing properties itself.

Only in case this default sizing behavior does not fulfill your requirements should you specify *width*- and *height*- properties explicitly.

### Don't Specify Percentage Heights

Do not specify *percentage heights* because the client might not be able to convert such relative heights into absolute values. Avoid setting the *height* property of UI-elements by default.

### Don't Specify Absolute Widths

Principally you should not specify *absolute* values for the *width*-property of your Web Dynpro UI-elements. Instead specify relative widths in *%.*

This rule is again based on the fact that the width of Web Dynpro UI elements could change because of localization changes, varying font sizes, lengths of contained fields, or because the application is embedded within a SAP NetWeaver Portal iView.

## Layout Rules

### Avoid Complex and Nested Layouts Wherever Possible

Avoid nested UI elements if they are not necessary. From a performance point of view it is often better to have one large *MatrixLayout* with column spans where necessary instead of a nested layout that includes transparent containers with *FlowLayout* or *GridLayout* and so on.

### Favor the Use of the *MatrixLayout*

Based on its flexibility and its ease of use you should favor the use of the *MatrixLayout*. This won't enhance the performance, but we think it is easier for the application developer to work with: You don't have to specify a column count and you can put as many controls into one row as you like. If you need a new row you simple set the property *layoutdata* of the first element of the new row to *MatrixHeadData*. Additionally, the *MatrixLayout* helps to achieve consistent layouts:

Other than the *GridLayout* or the *FlowLayout*, the *MatrixLayout* only allows some predefined values for the cell padding. Instead of the properties *paddingTop*, *paddingButtom*, *paddingLeft* and *paddingRight* it only offers the property *cellDesign* with the following predefined values (the standard *cellDesign* is **"rPad"**):

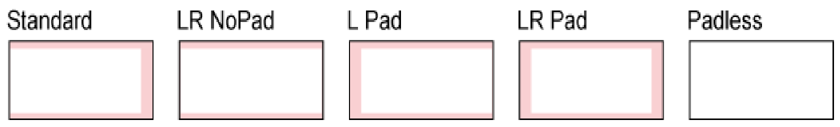


*Figure 3 – Predefined cell designs for the MatrixLayout*

Additionally the *MatrixLayout* provides the properties *hAlign*, *vAlign* to specify the horizontal and vertical alignment of the contained UI elements as well as the *vGutter*-property for horizontal distances between the cell contents.

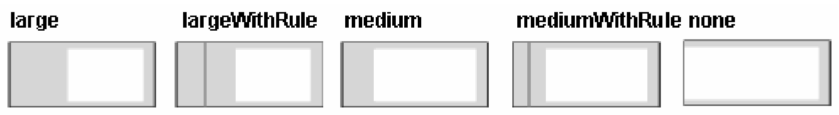


*Figure 4 – Predefined vertical gutters for the MatrixLayout*

### Avoid Using the *GridLayout*

The *GridLayout* should not be used in new view layouts any longer. This recommendation is based on the fact that the *GridLayout* cannot be laid out with *predefined* (like **"rPad"**), but with any combination of values (with CSS-sizes) for the different padding properties. Consequently it is more difficult to achieve a consistent setting of *cell padding* and *cell spacing* values within large applications (based on many view layouts in several UI components).

### When to Choose *RowLayout* Over *MatrixLayout*

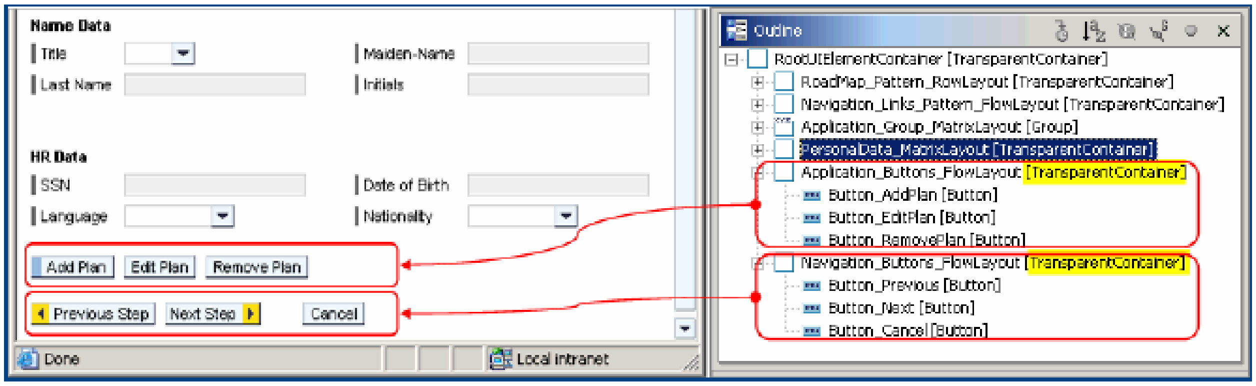
Choose the *RowLayout* over the *MatrixLayout* if you don't need horizontal alignment. When the *RowLayout* is implemented in an application, performance is better than if a *MatrixLayout* were used, but the layout flexibility is not compromised. For this reason, you should structure the view and container in horizontal areas as early as possible, using the *RowLayout*.

The *RowLayout* allows you to put an arbitrary number of elements in each row (it automatically uses a flow layout for each row). If you need a new row you simple set the property layout-data of the first element of the new row to *RowHeadData*. It is also a good idea to use the *RowLayout* if you have only one element per row as it is often used to arrange a screen vertically.

### Re-think the Usage of *TransparentContainers*

Re-think the usage of *TransparentContainer* UI elements inside container UI elements. Containers such as the *Group* UI element already allow specifying the layout of their content. Use the *layout*-property of the group instead of placing a *TransparentContainer* on it as top child and specify the layout there.

A reasonable usage of the *TransparentContainer* is to change the layout of inner UI elements compared with the layout of the embedding container. This is often done to arrange *Button* UI elements based on a *FlowLayout* whereas the embedding container UI element has a *MatrixLayout* (see figure 4):



```
RootUIElementContainer [TransparentContainer]
    RoadMap_Pattern_RowLayout [TransparentContainer]
    Navigation_Links_Pattern_FlowLayout [TransparentContainer]
    Application_Group_MatrixLayout [Group]
    PersonalData_MatrixLayout [TransparentContainer]
    Application_Buttons_FlowLayout [TransparentContainer]
        Button_AddPlan [Button]
        Button_EditPlan [Button]
        Button_RemovePlan [Button]
    Navigation_Buttons_FlowLayout [TransparentContainer]
        Button_Previous [Button]
        Button_Next [Button]
        Button_Cancel [Button]
```

*Figure 5 – Usage of TransparentContainer UI elements to layout buttons*

#### SAP NetWeaver 04s - Usage of TransparentContainers as Layout Containers

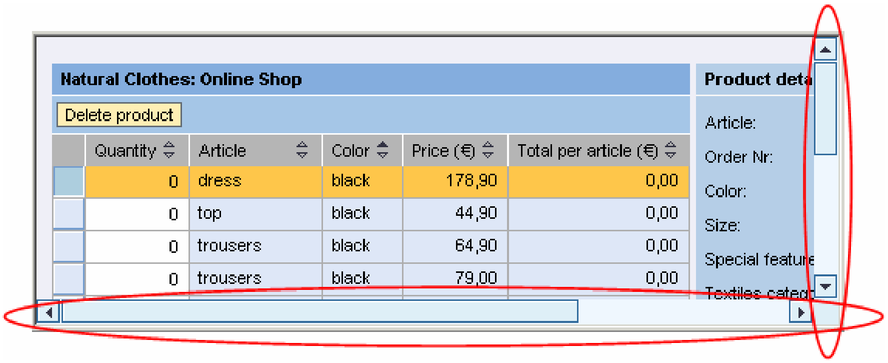
In SAP NetWeaver 04s you can replace *Group*-UI-Elements with *TransparentContainer*-UI-Elements of type *isLayoutContainer = false* to lay out label-field pairs and with header text on top.

## Scrolling Rules

### Use ScrollContainers Judiciously and Avoid Nesting Them

Principally your Web Dynpro UI should adhere to *immediacy*. Immediacy means that there should be *no need to scroll*. The user wants to see all information at a glance so that no further action (selection, submit, or scroll) should be required to see information.

Instead of adding your own custom scroll containers you should utilize the inbuilt scrolling capability of your Web Dynpro client. Within a browser client the horizontal and vertical scrollbars are made visible automatically as soon as the window gets smaller than the UI content:

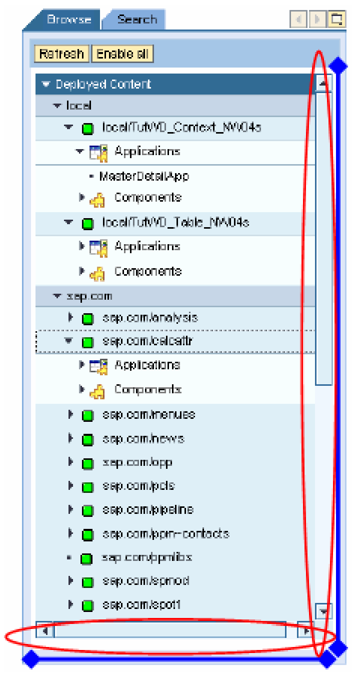


| Quantity | Article | Color | Price (€) | Total per article (€) |
|---|---|---|---|---|
| 0 | dress | black | 178,90 | 0,00 |
| 0 | top | black | 44,90 | 0,00 |
| 0 | trousers | black | 64,90 | 0,00 |
| 0 | trousers | black | 79,00 | 0,00 |

*Figure 6 – Generic scroll function of a browser client*

Consequently you should better design slim Web Dynpro user interfaces which do not require additional ScrollContainers to delimit “overcrowded” UI areas.

## When to Use ScrollContainers



It makes sense to embed potentially large UI areas within a *ScrollContainer*-UI-element so that the size this UI element is limited to a maximum height or width.

An example for this use-case is the Deployed-Content-Tree of the Web Dynpro Content Administrator application. To limit the size of the expandable *Tree*-UI-element it is embedded within a *ScrollContainer*-UI-element with absolute width and height values (in pixels).

### Note

- Avoid nested Scroll Containers
- Use ScrollContainers sparingly

*Figure 7 – Embedding a large tree within a ScrollContainer*

### SAP NW 04s – Apply the scrollingMode-property of Groups, Trays, and TransparentContainers

Within SAP NetWeaver 04s the container UI elements *Group, Tray* and *TransparentContainer* were extended by an additional *scrollingMode*-porperty (see `IWDScrollContainer`- and `WDScrollingMode`-API). Apply this new property instead of embedding inner scroll containers into *Groups*, *TransparentContainers,* or *Trays*.

### Set Width and Height of a *ScrollContainer* Explicitly

In case of embedding UI elements within a *ScrollContainer* you must set the *width*- and *height*- property explicitly. Otherwise the complete *ScrollContainer* may either collapse or its content may flow across the container borders (depending on the used client type).

Do not set the *width*- and *height*- property smaller than 32 pixels in order to avoid invisible scrollbars.

### Do Not Specify 100% Width for *ScrollContainers*

Due to a browser bug, the *ScrollContainer* UI element should not be set to 100% width.

### Do Not Use *ScrollContainers* in Tabs

To fulfill the *immediacy* principle within a Tab all its UI elements should be directly visible without the necessity to scroll. In case the size of the embedded UI elements gets too large, using a ScrollContainer seems to be a proper solution. Instead you should reconsider your UI design or *view composition* in terms of Web Dynpro. As soon as the content within a Tab UI element gets too large it is most often better to separate it to another view and to display it via navigation.

## Coding Rules

### Implement Fine-Grain UI Manipulations on Context but Not on UI Element Level

To maximize the rendering performance of your Web Dynpro application, implement all fine-grain UI manipulations on *context* instead of UI element level. This means do not call the setter-methods of UI element objects inside `wdDoModifyView()`directly but bind the related properties to the context and call the corresponding setter methods outside `wdDoModifyView():`

*Not*: `wdDoModifyView(){ … theField.setEnabled(true) … }`

*But*: `wdDoInit(){ … wdContext.currentFieldNode.setEnabled(true) … }`.

Otherwise Web Dynpro's caching mechanisms cannot speed up the rendering of view layouts.

The view controller's `wdDoModifyView`() method is designed for a special purpose: the creation of a UI tree or UI sub-tree at runtime in case it is not possible to declare the UI at design time. The method is neither intended for fine-grain UI manipulations nor for context manipulations. *Fine-grain* UI manipulations are achieved via means of data binding. Context manipulations are done in `wdDoInit()`, action event handlers or event handlers.

Unfortunately, the name `wdDoModifyView()` is somewhat misleading since the purpose of it is to create (and not to modify) a UI tree dynamically. While the usage of `wdDoModifyView()` in order to create a UI tree initially (`firstTime == true`) is not performance critical, modifying an existing UI tree (`firstTime == `**`false`**) is: Changing only one single attribute of a single UI element within `wdDoModifyView()` flags the complete view containing the control as dirty and the complete view must be completely re-rendered again – depending on the size and complexity of the view this can result in significant performance hits.

### Bind the UI Element Control Logic to the Context

If possible, bind all UI elements, including the dynamically created ones, against the view context. This way, subsequent changes to properties of the UI elements (*visibility*, *enabled* etc.) can be manipulated outside the `wdDoModifyView()` by manipulating the view context without invalidating the cached view layout data.

Especially bind UI element properties like *TreeNodeType.expanded*, *TabStrip.selectedTab* or *TableColumn.sortState* to properly typed context attributes.

Also specify the *readOnly*-property on context level, not on UI element level. The UI adapts automatically.

### Avoid Modification of View Layouts

As a consequence of the previous two rules you should avoid to modify view layouts repeatedly (`firstTime == false`). Only modify views during initialization (`firstTime == true`).

In case you cannot adapt a view layout to your need with data binding you should better create a separate, more specialized view layout instead and switch between them via navigation link.

## Related Content

1. Web Dynpro UI Element Reference (SAP NetWeaver 04)
2. Web Dynpro UI Element Reference (SAP NetWeaver 04s)
3. Tutorial on Using User Interfaces with Web Dynpro (23)

## Copyright