Play Sudoku - Using Table Cell Variants in Web Dynpro Java

Applies to:

Web Dynpro for Java applications for the SAP NetWeaver 7.0, SP 06 onwards.

Summary

With NetWeaver 7.0 some new table features were introduced in Web Dynpro for Java. This article, based on a sample project, explains three of them: *TableColumnGroup*, *TableStandardCell* and *TableSingleMarkableCell*.

Author: Stefanie Bacher

Company: SAP AG

Created on: 29 June 2007

Author Bio

Stefanie Bacher works as an information developer within the SAP NetWeaver Product Management User Interaction Team. She focuses on knowledge distribution of Web Dynpro for Java.

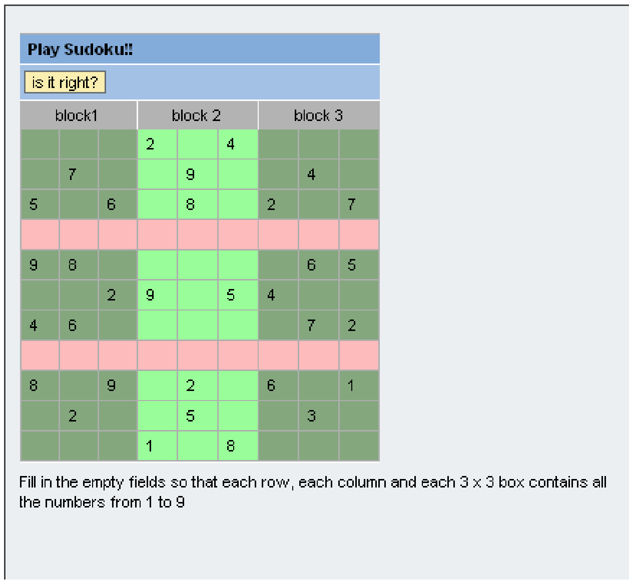
Table of Contents

Using Cell Variants in Web Dynpro Table UI element

Running the Sample Application

Download the sample project here: WDJ Table Cell Variants sample project.zip

Unzip and import the project, deploy and run the application:



block1			block 2			block 3		
			2		4			
	7			9			4	
5		6		8		2		7
9	8						6	5
		2	9		5	4		
4	6						7	2
8		9		2		6		1
	2			5			3	
			1		8			

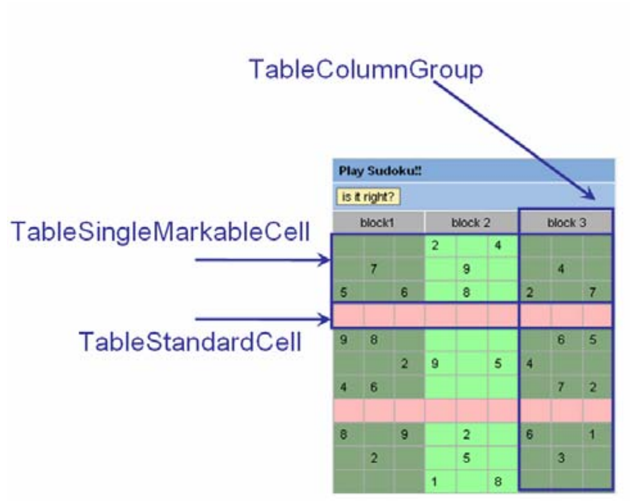
Fill in the empty fields so that each row, each column and each 3 x 3 box contains all the numbers from 1 to 9

The table, you see, is built up using the following view elements:

TableColumnGroup.
This allows to group table columns under a common header.

TableStandardCell.
Using this cell variant you can define different designs for cells and even choose another cell editor. This can for example be used to define a row where input fields allow the user to enter new data, whereas the other rows just show the existing data in display mode.

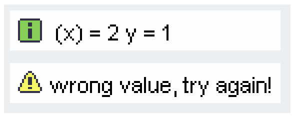
TableSingleMarkableCell
Allows you to operate on a single cell instead of the row, in this case it is used to check whether the entered number matches the data provided in the solution.



block1			block 2			block 3		
			2		4			
	7			9			4	
5		6		8		2		7
9	8						6	5
		2	9		5	4		
4	6						7	2
8		9		2		6		1
	2			5			3	
			1		8			

After selecting a single cell, you can enter a number and then check by clicking the "*is it right?*" button.

Two messages will be displayed:



(x) = 2 y = 1

wrong value, try again!

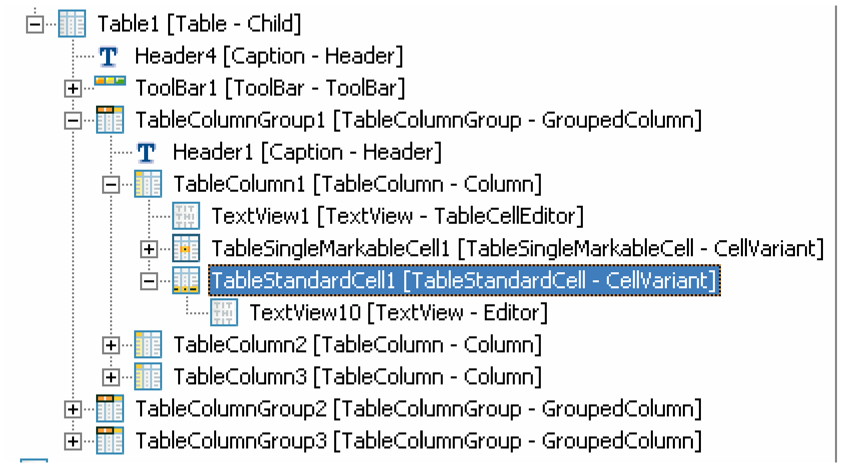
What does that mean?

The first message is for developers: It tells you the coordinates of the cell you operated on

The second message is for players and gives you the information whether the entered value matches the solution.

Now, what has to be done to achieve this?

Creating the Table

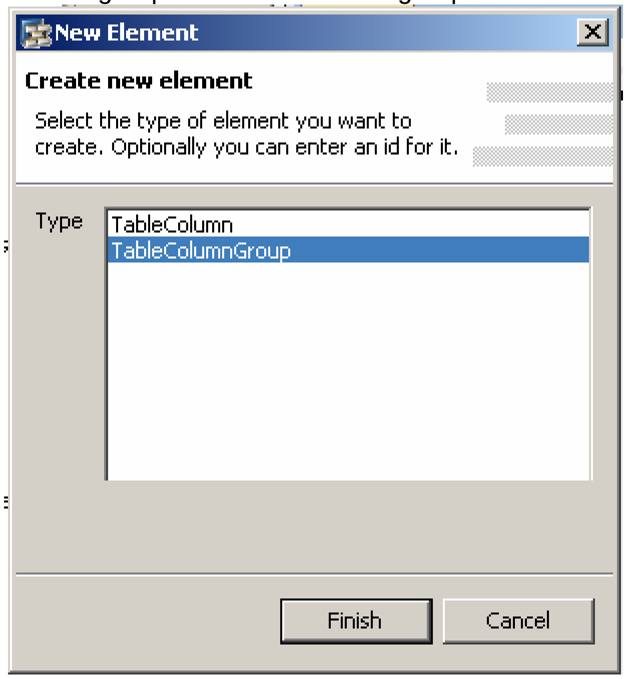


```
Table1 [Table - Child]
    Header4 [Caption - Header]
    ToolBar1 [ToolBar - ToolBar]
    TableColumnGroup1 [TableColumnGroup - GroupedColumn]
        Header1 [Caption - Header]
        TableColumn1 [TableColumn - Column]
            TextView1 [TextView - TableCellEditor]
            TableSingleMarkableCell1 [TableSingleMarkableCell - CellVariant]
            TableStandardCell1 [TableStandardCell - CellVariant]
                TextView10 [TextView - Editor]
        TableColumn2 [TableColumn - Column]
        TableColumn3 [TableColumn - Column]
    TableColumnGroup2 [TableColumnGroup - GroupedColumn]
    TableColumnGroup3 [TableColumnGroup - GroupedColumn]
```

The figure above shows the structure of the Sudoku table:

First of all, the Table's `selectionMode` property has to be set to `none;` otherwise the *TableSingleMarkableCell* won't work.

To define the table columns in groups, select the table, open the context menu, choose *Insert Grouped Column* and then *TableColumnGroup*. Select this *TableColumnGroup*, open the context menu, choose *Insert Grouped Column* again and select *TableColumn*. Repeat these steps until the table contains three table column groups and each column group contains three table columns.


New Element
Create new element
Select the type of element you want to create. Optionally you can enter an id for it.
Type
TableColumn
TableColumnGroup
Finish
Cancel

A *TableColumnGroup* contains per default a *Header*, whose text property allows you to specify a title stretched over all contained columns.

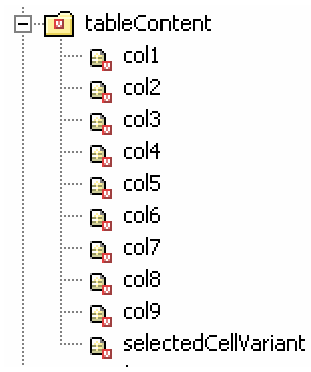
Into each TableColumn insert two cell variants: A TableStandardCell and a TableSingleMarkableCell. Each cell variant needs a TableCellEditor. For the TableStandardCell choose TextViews, for the TableSingleMarkableCell InputFields.

In this sample *TableStandardCell* is used to separate the blocks by bars in the forth and in the eighth table row. For every *TableStandardCell* the same `variantKey` is defined, in this example variant2 and – by the way – the same `cellDesign` property is set: `badvalue_light`.

In the same way all *TableSingleMarkableCells* have the same `variantKey`, here: `variant1`.

Setting the Cell Variants Dynamically

The view controller context contains a node called *tableContent*. Beside the context attributes called *col1*, *col2*… (they are needed for the data to be displayed), there is a context attribute: *selectedCellVariant* of type `string`.



Using this context attribute you can define by coding in which row which cell variant is displayed. To do this, you need some lines of code. Inside a for loop, where the node elements for the table are created, the respective cell variants are set:

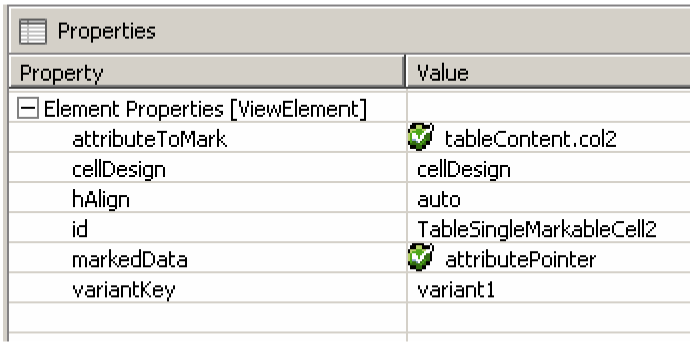
```
...
switch (i) {
    case 3 :
        elem.setSelectedCellVariant("variant2");
        break;
    case 7 :
        elem.setSelectedCellVariant("variant2");
        break;
    default :
        elem.setSelectedCellVariant("variant1");

}

...
```

Now the relevant cell variants are displayed at the right position.

Retrieve the x and y Coordinates of a Cell

A *TableSingleMarkableCell* has two properties that need to be defined to get things to work:

- `attributeToMark`
- `markedData`

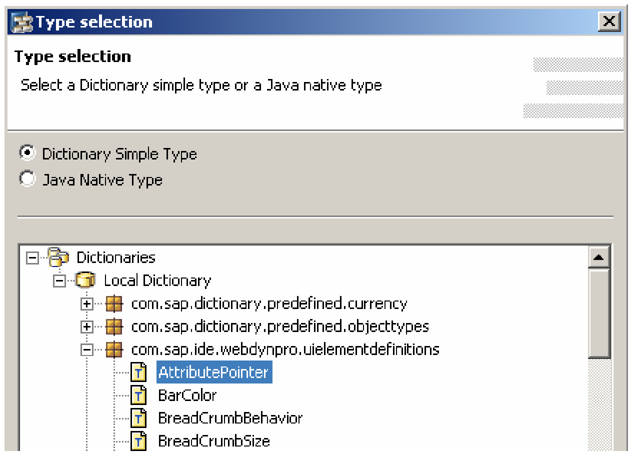

Properties

Property	Value
Element Properties [ViewElement]	
attributeToMark	tableContent.col2
cellDesign	cellDesign
hAlign	auto
id	TableSingleMarkableCell2
markedData	attributePointer
variantKey	variant1

In the properties of each *TableSingleMarkableCell*, the `attributeToMark` property has to be set to that context attribute, which stores the data of this table column where the *TableSingleMarkableCell* is located. To give an example: The *TableSingleMarkableCell2* that is inserted into the second *TableColumn* has to be bound to the context attribute *col2*.

The `markedData` property – again for each *TableSingleMarkableCell* - has to be bound to a context attribute of type *AttributePointer*.

To define this type for a context attribute, click on the three dots ... on the right hand side of the `type` property row (supposed you are in the *Property view* of your context) and select *Dictionary SimpleType* in the wizard that opens. Open the package *com.sap.ide.webdyn pro.uielementdefinitions* and select *AttributePointer*. If a single cell is selected, a pointer to this cell is stored in the context element *attributePointer*.


Type selection

Type selection

Select a Dictionary simple type or a Java native type

[✓ Dictionary Simple Type]
Java Native Type

Dictionaries
Local Dictionary
com.sap.dictionary.predefined.currency
com.sap.dictionary.predefined.objecttypes
com.sap.ide.webdynpro.uielementdefinitions
AttributePointer
BarColor
BreadCrumbBehavior
BreadCrumbSize

In this example this is done in an action handler called `onActiontestASingleValue`:

To retrieve this pointer, you need the following coding:

```
    IWDAttributePointer attPointer =
    wdContext.currentContextElement().getAttributePointer();
```

To identify the x coordinate, retrieve the column name and set the respective value for x:

```
String colName = attPointer.getAttributeInfo().getName();
if (colName.equals("col1"))
    x = 1;
```

To identify the y-coordinate, retrieve the index of the node element:

```
int y = attPointer.getNodeElement().index()+1;
```

Related Content

Online Documentation on Sap Help Portal: Cell Variants

Copyright