Implementing a Table with Sorting and Filtering Capabilities

Applies to:

Web Dynpro for Java applications for SAP enhancement package 1 for SAP NetWeaver Composition Environment 7.1. For more information, visit the User Interface Technology homepage.

Summary

This tutorial demonstrates the basic concept of the table UI element and how to implement additional functionality such as sorting and filtering, provided by Java classes that interact with the Web Dynpro framework.

Author: Martin Clabunde, thanks to Chris Whealy for helpful contribution.

Company: SAP AG

Created on: 30th September, 2008

Author Bio

Martin Clabunde is studying computer science at the Univerity of Applied Science Worms and works as a working student in Product Management for SAP NetWeaver User Interaction Technology.

Table of Contents

Prologue

Prerequisites

- o Systems, installed applications, and authorizations
- o The SAP NetWeaver Developer Studio is installed on your computer.
- o You have access to the SAP J2EE Engine.

- You have acquired some basic experience with Web Dynpro applications - for example, by working through the Welcome Quick start Guide (Web Dynpro Java for Newbies).
- Basic knowledge of Java would be an advantage.

Details

Level of complexity: Beginner

Time required for completion: 60 min.

Sample Project

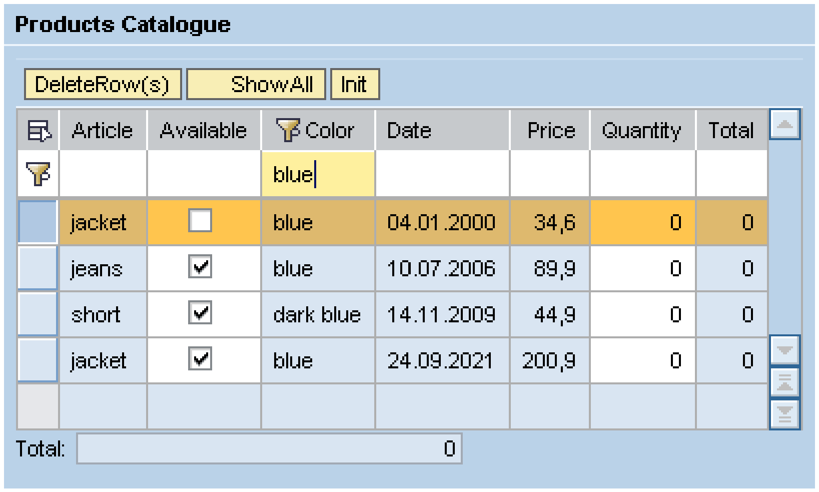
The ready-use Web Dynpro DC including the TableSorter and TableFilter Java classes can be downloaded as a zip archive from here.

Introduction

The tutorial focuses on the Table UI element. Furthermore, it shows you how to add custom functionality in Web Dynpro by using Java classes. The table UI element supports sorting and filtering events, so this tutorial will show you how to implement this logic using custom written Java classes.

First, you will see how fast and simple it is to display context information using the table UI element. Then, we will implement methods to add, delete, and copy rows of the table. Lastly, the most important scenario is sorting and filtering of table data.

The two classes needed for sorting and filtering will be bound to the relevant context attributes, instantiated and associated with the table UI via its event properties. Additional functions such as “Delete row” and “Initialize” are supplied by buttons in the table’s toolbar.



	Article	Available	Color	Date	Price	Quantity	Total
			blue				
	jacket	☐	blue	04.01.2000	34,6	0	0
	jeans	✓	blue	10.07.2006	89,9	0	0
	short	✓	dark blue	14.11.2009	44,9	0	0
	jacket	✓	blue	24.09.2021	200,9	0	0

Total: 0

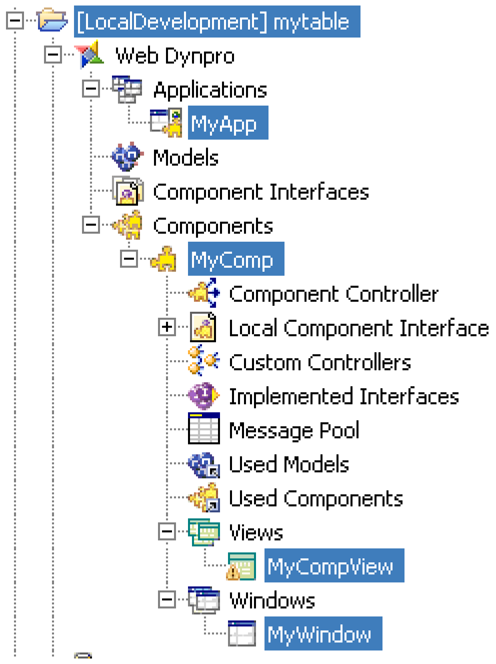
In this tutorial, you will learn how to:

- Create the context of the component controller and its structure to the view controller
- Create a table UI element and map it to the view controller’s context
- Implement table sorting functionality
- Implement table row filtering functionality
- Implement a function to delete one or more rows
- Implement a function for calculating the total for selected articles and an overall total

Since the Java classes that implement the sorting and filtering functionality are independent of the Web Dynpro Framework, they must be written carefully. These classes should not make any assumptions about the environment from which they are invoked.

Create a new Web Dynpro DC

Create a new Web Dynpro DC with all it´s necessary units. Name the DC **mytable** the Application **MyApp** and the Component **MyComp**. Leave the Default Window and Views option selected in order to get them applied by default. Then rebuild the project so that the various Java source files are created.



Creating the Context for the Table Data

To provide a table with data, we must first store that data in the context of the *Component Controller*.

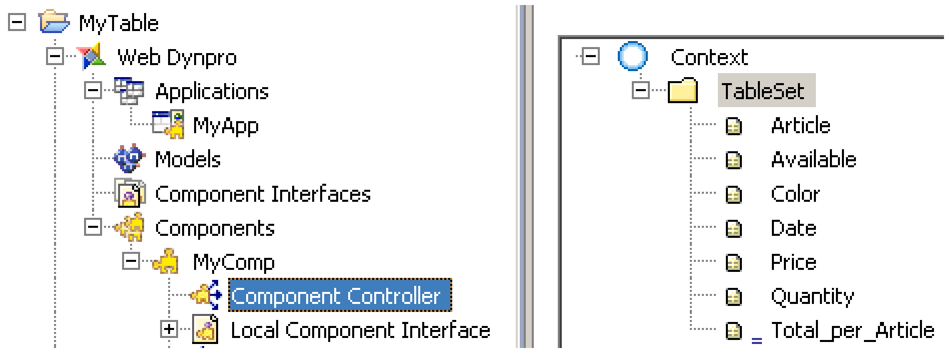
1. Add a new Node to the context root node and name it **TableSet**. The *Collection Cardinality* of the *TableSet* node should be left at the default value of **[0...n]**.

2. Change the Selection Cardinality to **[0...n]**. Both properties can be found in the *Properties* tab in the lower screen section.

3. Now, manually add attributes to this node and set the data types according to the table below. Since *Total_per_Article* should be a *calculated* attribute, select the *calculated* checkbox to true. This causes the NWDS to generate the appropriate Accessor/Mutator methods. It is important to set the *Read-only* property in the *Properties* tab to **false** as the filter class needs to update the calculated attribute.

Attribute Name	Type	Calculated	Read-only
Article	<string>	no	false
Available	<boolean>	no	false
Color	<string>	no	false
Date	<date>	no	false
Price	<decimal>	no	false
Quantity	<integer>	no	false
Total_per_Article	<decimal>	yes	false

The context of the Component Controller now should look like this:

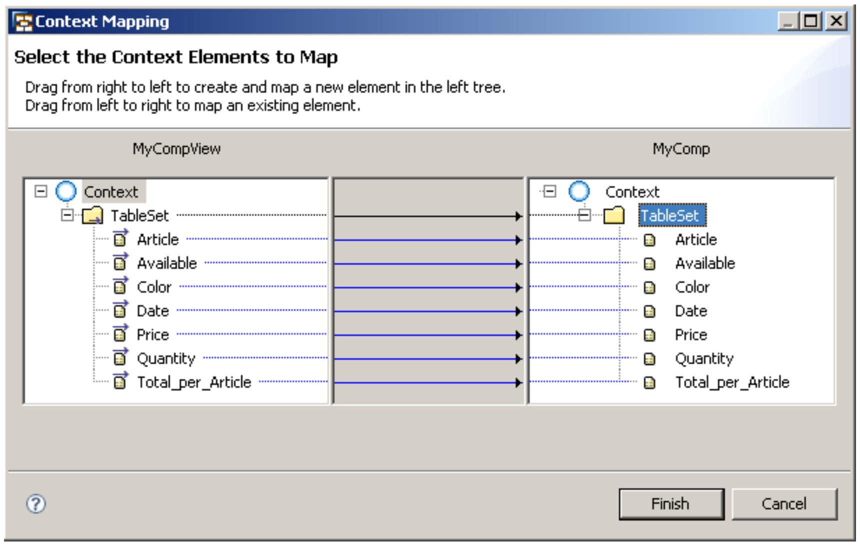


Mapping the View Context onto the Component Controller Context

In order to make the data held in the context of the Component Controller available to the View Controller, we need to tell the View Controller *MyCompView* that it should use the Component Controller as a data source. Once this has been done, the data in the Component Controller's context is available to the View Controller's context through a technique known as *Context Mapping*.

1. Double-click the component *MyComp* and create a data link between the *MyCompView* icon and the Component Controller icon. The causes the *component controller* to be added to the view controller's list of *Required Controllers*.
2. By using drag and drop, drag the *TableSet* node from the Component Controller context on the right (*MyComp*) to the left context root node (*MyCompView*). Then unselect the *TableSet* node checkbox and select it again in order to select all attributes. Confirm with *Finish*.

This wizard has now duplicated the *TableSet* node found in the Component Controller in the context of the View Controller and created a mapping relationship. Now, when you read and write data to and from the *TableSet* node in the View Controller, you are actually reading and writing data that lives in the Component Controller.

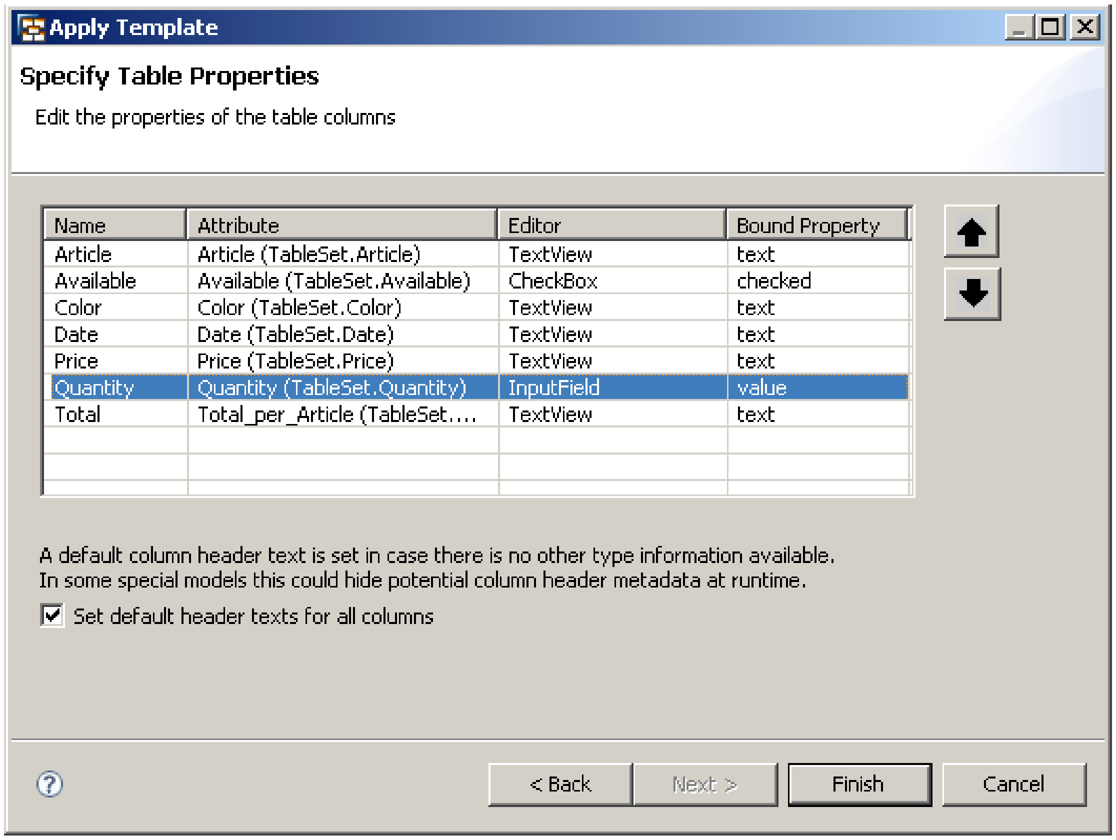


Designing the View Layout

Now that the Component Controller data is available to the View Controller, we can start to add UI elements to the View Controller's layout to display this information on the screen.

To present all the articles on the browser screen, we just have to add some UI elements to the View Controller's layout.

1. Open the *MyCompView* View Editor by double-clicking it or via the context menu and switch to the *Layout* tab.
2. In the Outline View delete the *DefaultTextView* element.
3. Right-click on the *Root Element* of the *Outline* View and choose insert *child…* In the next window choose the UI element *Group* and set its *Group_Header* element *text* property to **Products Catalogue** in its *Properties* tab.
4. Right-click on the UI element *Group* and choose *Apply Template*.
5. Select the *Table* icon and press *Next*.
6. Select the *TableSet* node checkbox. All attributes are selected automatically and press *Next*.
7. In the Specify *Table Properties* window change the editor of the value attribute *Quantity* from *TextView* to *InputField* and confirm with *Finish*.



Specify Table Properties

Edit the properties of the table columns

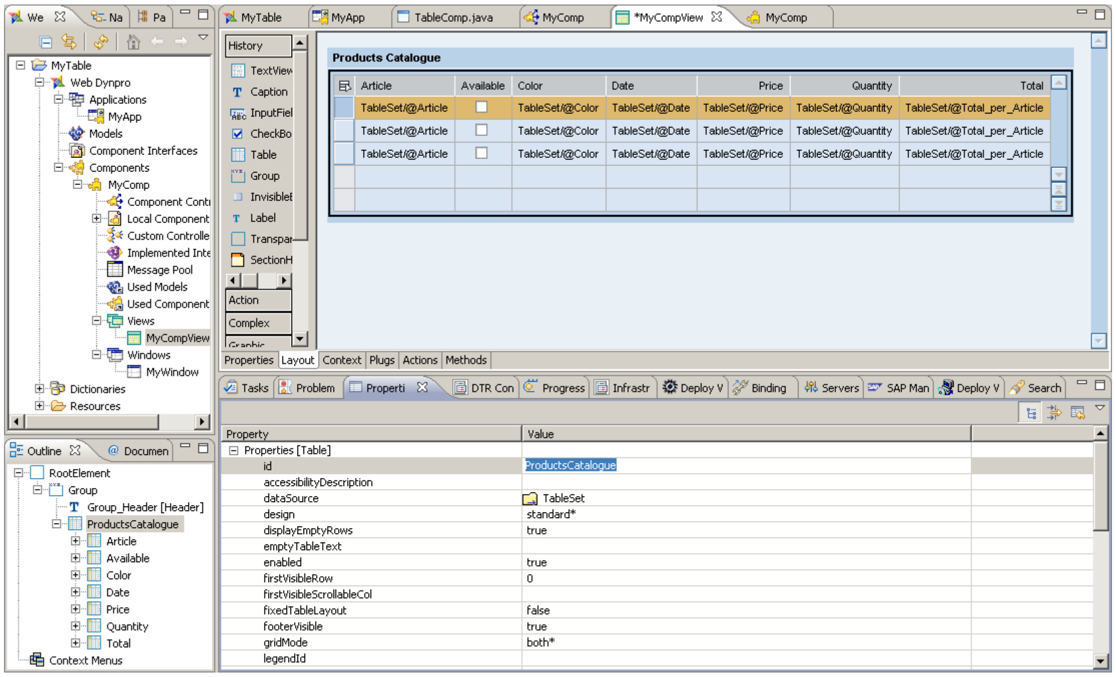
Name	Attribute	Editor	Bound Property
Article	Article (TableSet.Article)	TextView	text
Available	Available (TableSet.Available)	CheckBox	checked
Color	Color (TableSet.Color)	TextView	text
Date	Date (TableSet.Date)	TextView	text
Price	Price (TableSet.Price)	TextView	text
Quantity	Quantity (TableSet.Quantity)	InputField	value
Total	Total_per_Article (TableSet.…	TextView	text

A default column header text is set in case there is no other type information available.
In some special models this could hide potential column header metadata at runtime.

☑ Set default header texts for all columns

8. Change the Table id to **ProductsCatalogue** in the Properties tab of the *Table* element. This value then becomes the runtime name for the UI element instance.

The Table UI and its appropriate data columns:



Products Catalogue

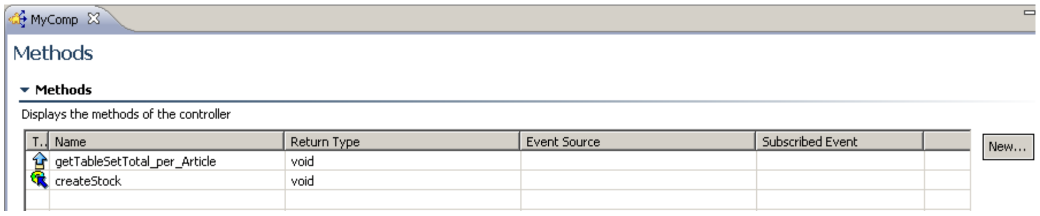
Article	Available	Color	Date	Price	Quantity	Total
TableSet/@Article	☐	TableSet/@Color	TableSet/@Date	TableSet/@Price	TableSet/@Quantity	TableSet/@Total_per_Article
TableSet/@Article	☐	TableSet/@Color	TableSet/@Date	TableSet/@Price	TableSet/@Quantity	TableSet/@Total_per_Article
TableSet/@Article	☐	TableSet/@Color	TableSet/@Date	TableSet/@Price	TableSet/@Quantity	TableSet/@Total_per_Article

Property	Value
Properties [Table]	
id	ProductsCatalogue
accessibilityDescription	
dataSource	TableSet
design	standard*
displayEmptyRows	true
emptyTableText	
enabled	true
firstVisibleRow	0
firstVisibleScrollableCol	
fixedTableLayout	false
footerVisible	true
gridMode	both*
legendId	

Providing some data

In order for the table to display any data, we must first some code to generate that data.

1. Double-click on the *Component Controller*
2. Switch to the *Methods* tab and press the *New…* button. Add a method (as opposed to an event handler) and give it the name **createStock**. Confirm with *Finish*.



Methods

Displays the methods of the controller

T..	Name	Return Type	Event Source	Subscribed Event
	getTableSetTotal_per_Article	void		
	createStock	void		

Note: Make sure that in the menu *Project > Build Automatically* is marked and save subsequently. Otherwise you have to (re)build the project manually each time a declarative change is made such as the addition of a new method.

3. Select the *createStock()* method and press the *F3 key* or navigate to its method implementation via the context menu.
4. Between the `//@@begin createStock()` and `//@@end` tags insert following code:

```
Calendar myCalendar = Calendar.getInstance();
myCalendar.setLenient(false);
java.sql.Date tmpDate = new java.sql.Date(0);
int stockLen = STOCK.length;
int m = 1, d = 1,y = 1980;

for ( int i = 0; i < stockLen; i++ ) {
   ITableSetElement product = wdContext.createAnddAddTableSetElement();
   product.setAvailable((i%6 == 0) ? false : true);
```

```
    product.setArticle(STOCK[i][1]);
    product.setColor(STOCK[i][2]);
    product.setPrice( new BigDecimal(STOCK[i][3]));
    d = ( i % 2 == 0 ) ? (i+4)%30 : d;
    m = ( i % 2 == 0 ) ? i%12 : (i-1)%12;
    y = ( i % 3 == 0 ) ? 2000+i : y;
    myCalendar.set(y,m,d);
    tmpDate.setTime(myCalendar.getTimeInMillis());
    product.setDate(tmpDate.valueOf(tmpDate.toString()));
}
```

5. *Organize Imports* (*Ctrl+Shift+o*), choose *myapp.comp.wdp.**IPublicMyComp**.ITableSetElement* and confirm with *Finish*.

6. Between the `//@@begin others` and `//@@end` tags, which can be found at the end of the file, insert:

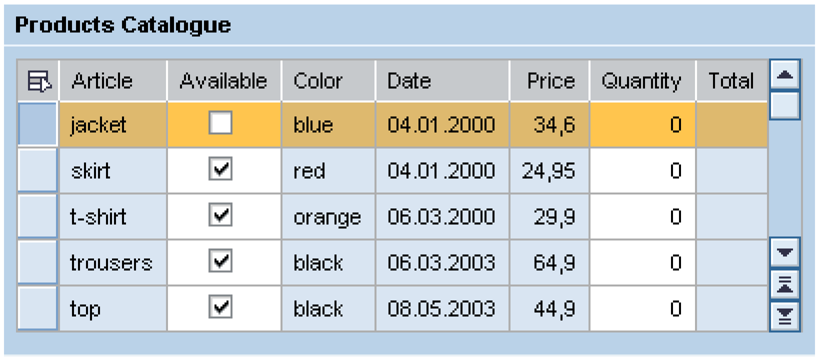
```
private static final String STOCK[][] = {
          {"0","jacket","blue","34.60"},
          {"1","skirt","red","24.95"},
          {"2","t-shirt","orange","29.90"},
          {"3","trousers","black","64.90"},
          {"4","top","black","44.90"},
          {"5","dress","colored","78.90"},
          {"6","blouse","white","35.50"},
          {"7","jeans","blue","89.90"},
          {"8","pullover","red","69.00"},
          {"9","sweatshirt","green","61.60"},
          {"10","polo shirt","yellow","14.65"},
          {"11","short","dark blue","44.90"},
          {"12","blouse","white","49.90"},
          {"13","skirt","white","55.00"},
          {"14","pullover","white","127.00"},
          {"15","dress","black","178.90"},
          {"16","top","red","54.00"},
          {"17","trousers","black","79.00"},
          {"18","t-shirt","red","45.60"},
          {"19","jacket","white","55.80"},
          {"20","pullover","white","130.90"},
          {"21","jacket","blue","200.90"},
          {"22","skirt","brown","89.90"},
          {"23","alpaca pullover","brown","230.00"},
          {"24","lambswool pullover","yellow","130.00"}
};
```

7. Between the `//@@begin wdDoInit()` and `//@@end` tags of the `wdDoInit()` method insert:

```
createStock();
```

8. *Save* the project, *Deploy new Archive and run*.

You are able to display the table with the *STOCK* items yet.



Products Catalogue

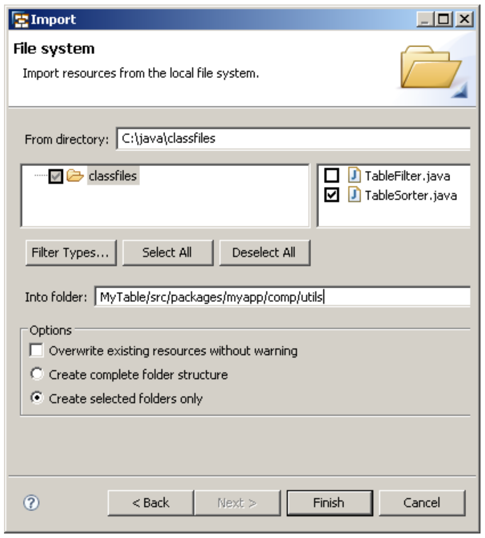
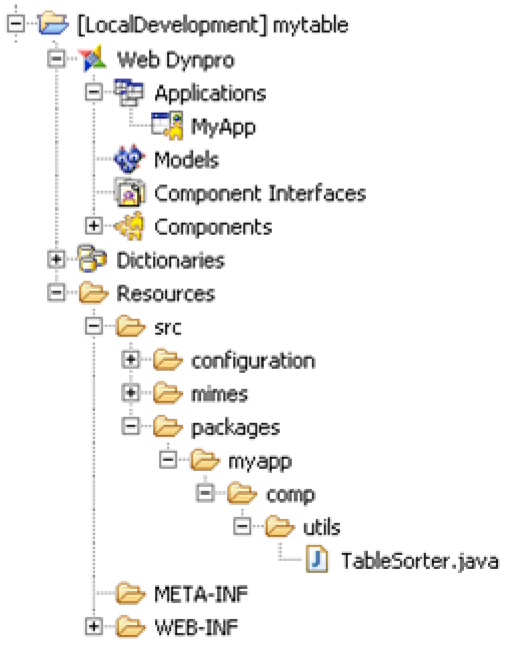
	Article	Available	Color	Date	Price	Quantity	Total
	jacket	☐	blue	04.01.2000	34,6	0	
	skirt	☑	red	04.01.2000	24,95	0	
	t-shirt	☑	orange	06.03.2000	29,9	0	
	trousers	☑	black	06.03.2003	64,9	0	
	top	☑	black	08.05.2003	44,9	0	

Implementing Sorting

Since the Web Dynpro Framework does not provide any sorting logic as standard, this must be implemented by the application developer and can be handled by a separate java class. In this scenario we will make use of the `TableSorter.java` class. This has been prewritten, so it should be imported into to the project structure and an instance assigned to a context attribute of its own type.

Importing the Java TableSorter class

1. Select *Import* from file menu.
2. Choose *General* > *File System* and click *Next.*
3. Specify the folder where the Java class (*TableSorter.java*) is stored and select it by selecting the corresponding checkbox.
4. Now specify where to store the class in the current project structure by using the second browse button '*Into Folder*'. Browse the project structure as follows: *mytable* > *src* > *packages* > ..... > *myapp* > *comp* and confirm with *OK*.
5. Manually append to this path a new folder and name it **utils**, so that the path specified describes following location: *mytable/src/packages/.../myapp/comp/utils* and confirm with *Finish*. If you choose a different location, you need to change the package included in the java class accordingly.

Import Window:	Resulting project structure:
	

Import window:

File system

Import resources from the local file system.

From directory: C:\java\classfiles

☑ classfiles

☐ TableFilter.java
☑ TableSorter.java

Into folder: MyTable/src/packages/myapp/comp/utils

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

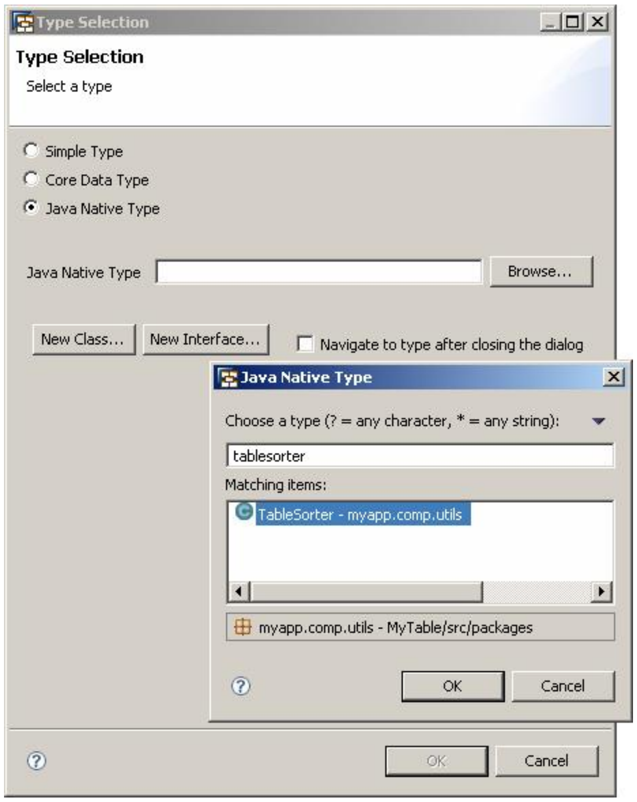
Resulting project structure:

```
[LocalDevelopment] mytable
  Web Dynpro
    Applications
      MyApp
    Models
    Component Interfaces
    Components
  Dictionaries
  Resources
    src
      configuration
      mimes
      packages
        myapp
          comp
            utils
              TableSorter.java
    META-INF
    WEB-INF
```

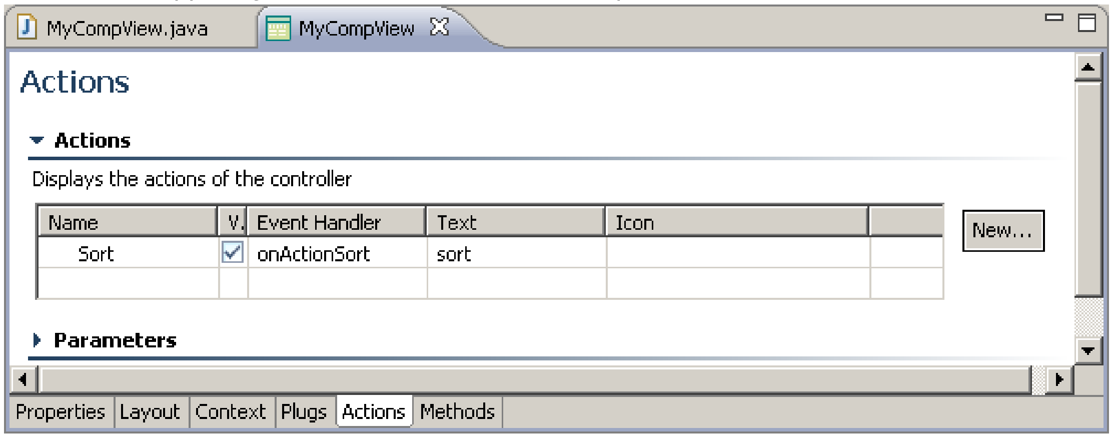
Enhancing the Context

We need to store the *TableSorter* instance in a context attribute to be able to access it at runtime. Each time the user clicks on a table column header, we need to implement the *OnAction* event in order to get the table sorted correctly.

1. Double-click the Component Controller and switch to the Context tab
2. Add a manual attribute to the context root node. Call it **TableSorter** and browse for its *Java Native Type* by typing its class name *TableSorter* into the filter input field. The class should be found and displayed in the lower area *Matching Items*. If more than one item of the same class name was found, select the one matching your project´s structure path, e.g. *myapp.comp.utils*, click *OK* and confirm with *Finish*.



3. Double-click *MyComp* in the project structure, double-click the data link between the *MyCompView* and *Component Controller* icons and map the *TableSorter* to *MyCompView*.
4. Double-click *MyCompView* and add a new *Action*. Switch to the *Action* tab and press the *New…* button in the upper right. Name the action **Sort** and press *Finish*.



Name	V.	Event Handler	Text	Icon
Sort	✓	onActionSort	sort	

5. Open its method implementation by pressing the *F3 Key* or via context menu and add following code between the `//@@begin onActionSort(ServerEvent)` and `//@@end` tags:

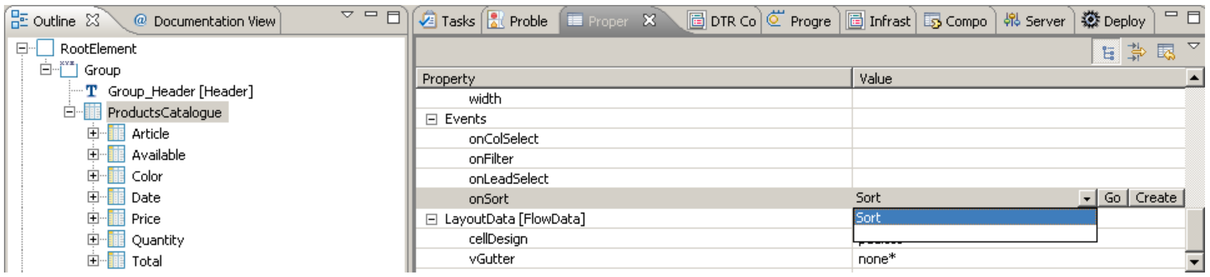
```
wdContext.currentContextElement().getTableSorter().sort( wdEvent,
       wdContext.nodeTableSet() );
```

6. Switch to the *wdDoModifyView()* method and insert following code between the `//@@begin wdDoModifyView` and `//@@end` tags:

```
if (firstTime) {
  IWDTable table = (IWDTable)view.getElement("ProductsCatalogue");
  wdContext.currentContextElement().setTableSorter( new
      TableSorter(table, wdThis.wdGetSortAction(), null));
}
```

7. *Organize Imports* (*Ctrl+Shift+o*),

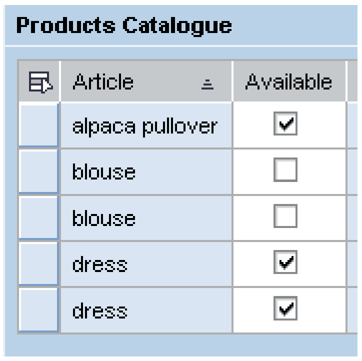
8. Switch to the *Layout* tab of *MyCompView* and select the *ProductsCatalogue* Table UI element from the *Outline* view in the bottom left. Select the *Properties* tab and associate the table's *onSort* event with the Sort action we have just created.



Property	Value
width	
Events	
onColSelect	
onFilter	
onLeadSelect	
onSort	Sort
LayoutData [FlowData]	
cellDesign	
vGutter	none*

9. *Save*, *Deploy new Archive and Run*

Now after you click on a column header, a sort order icon appears on the sorted column.



Products Catalogue

Article	Available
alpaca pullover	✓
blouse	☐
blouse	☐
dress	✓
dress	✓

Implementing Filtering

So far, displaying and sorting of table data isn´t as difficult as it may at first have appeared, but in order to be able to *filter* table data, we need to make some changes to the context. The steps we will follow are nearly the same as those required to implement the sort capability.

In order to apply filtering to a Table, a total of three context nodes are required: One for the table data (which we already have), one to hold the filter values and one to hold the subset of rows after filtering has been applied. All three nodes need to have the same structure and should be located directly under the root node.

Filtering capabilities

The table filter has the following capabilities:

- **Strings:** if the filter value contains simply the letter "a", it treats this as if you had entered "*a*" and will return any fields that contain an "a" in any position. The search is **not** case sensitive!

  It also accepts the operators "**#**" (for exclude) or "**=**" (for include) in the first position.

- **Numeric values, dates and times:** The filter will search either for the exact value, or if the following operators are used, ranges, inclusion and exclusion can be specified.

  If you use the include "**=**" or exclude "**#**" operators, then these should always appear in the first position.

  For ranges:

"~100"	gives all values up to and including 100
"1~100"	gives all values between 1 and 100 inclusive
"100~"	gives all values greater than or equal to 100

- **Boolean values:** Booleans can be filtered using the include "**=**" or the exclude "**#**" operators in the first position.

Importing the Java TableFilter class

The process to import the TableFilter.java class is exactly the same as was used for the *TableSorter.java* class. Only in this case, the *utils* folder already exists in the project structure and we will just copy the class to this location.

1. Select Import from the file menu, choose *General* > *File System* and click *Next*.
2. Specify the folder where the Java class (*TableFilter.java*) is located and select it.
3. Now specify where to store the class file in the current project structure by using the second browse button '*Into Folder*'. Browse the project structure as follows:

   mytable > *src* > *packages* > *...* > *myapp* > *comp* > *utils*, click *OK* and confirm with *Finish*.

Enhancing the Context

By default, there is no concept of data normalization within a Web Dynpro context node; therefore a context node has no concept equivalent to that of a "database key". So, in order to identify a data record unambiguously, the *TableSet* context node needs to be extended so that the filter has a '*unique identifier*' for each row.

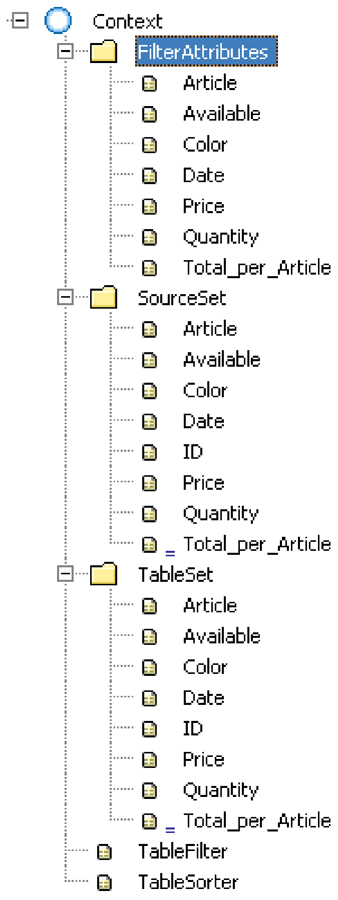
1. Double-click the *Component Controller* and switch to the context tab.
2. Add a new attribute called **ID** of type *string* to the *TableSet* Node.
3. Copy the entire *TableSet* node to the clipboard by choosing the *copy* function from the context menu, paste it directly to the root node and rename it from *TableSet_1* to **SourceSet** by pressing the *F2 key* or via context menu. Alternatively you can add a new node to the root context node, name it **SourceSet** and add all the attributes as in *TableSet* exists, one by one.
4. Set the *Read-only* property of the *Total_per_Article* attribute in *TableSet* node and *SourceSet* node to **false**.
5. Directly under the context root node, add a new *Java Native Type* attribute named **TableFilter**. Set the data type of this attribute to *TableFilter* in the same way you did for the *TableSorter* attribute.
6. Create a new node under the context root node called **FilterAttributes** and following attributes shown in the table below. Set the node´s *collection cardinality* property to **[1..1]**.

   Note that all attributes are of type *string*. These attributes represent the filterable columns of the table.

Attribute Name	Type	Calculated	Read-only
Article	<string>	no	false
Available	<string>	no	false
Color	<string>	no	false
Date	<string>	no	false
Price	<string>	no	false
Quantity	<string>	no	false
Total_per_Article	<string>	no	false

Notice that the **ID** attribute is not required in this context node.

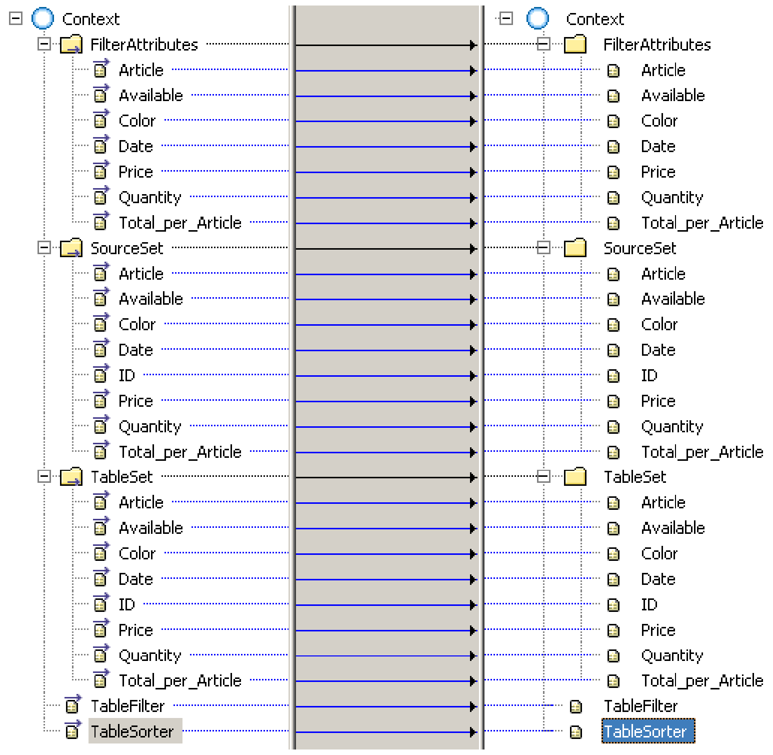
The context should now look like this:



```
Context
  FilterAttributes
    Article
    Available
    Color
    Date
    Price
    Quantity
    Total_per_Article
  SourceSet
    Article
    Available
    Color
    Date
    ID
    Price
    Quantity
    Total_per_Article
  TableSet
    Article
    Available
    Color
    Date
    ID
    Price
    Quantity
    Total_per_Article
  TableFilter
  TableSorter
```

Mapping the context again

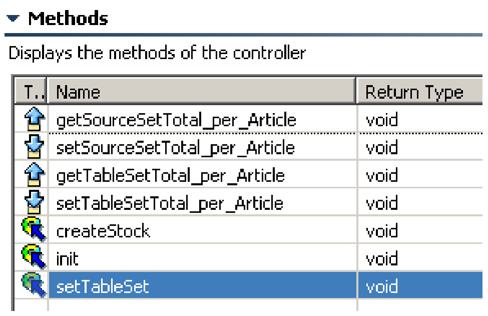
1. Double-click the *MyComp* component and then double-click the data link between *MyCompView* and the *Component Controller* icons to open the *Context Mapping* window.
2. Drag the *FilterAttributes* and *SourceSet* nodes as well as the *TableFilter* attribute across from the component controller's context, and drop them on the root node of the view controller's context.
3. Since we have added a new attribute to the *TableSet* node, this also should be mapped to the view controller. Therefore, without leaving the Context Mapping wizard, right-click on the *TableSet* node of *MyCompView* and select "Edit Context Mapping".
4. Select the **ID** field and press *Finish*, then press *Finish* again.



Methods and Actions

Component Controller

1. Double-click the *Component Controller* and switch to the *Methods* tab.
2. Add two new Methods: **init**, **setTableSet**.


Methods

Displays the methods of the controller

T..	Name	Return Type
	getSourceSetTotal_per_Article	void
	setSourceSetTotal_per_Article	void
	getTableSetTotal_per_Article	void
	setTableSetTotal_per_Article	void
	createStock	void
	init	void
	setTableSet	void

3. *Save* and *rebuild* the project.

4. Select the *init()* Method and jump to its implementation by pressing the *F3 key* and insert following code between the `//@@begin init()` and `//@@end` tags:

```
wdContext.nodeSourceSet().invalidate();
createStock();
setTableSet();
```

5. Go to the *setTableSet()* method and insert the code between the `//@@begin setTableSet()` and `//@@end` tags:

```
WDCopyService.copyElements(wdContext.nodeSourceSet(),
         wdContext.nodeTableSet());
```

6. Go to the *wdDoInit()* method and replace the *createStock()* method call with:

```
init();
```

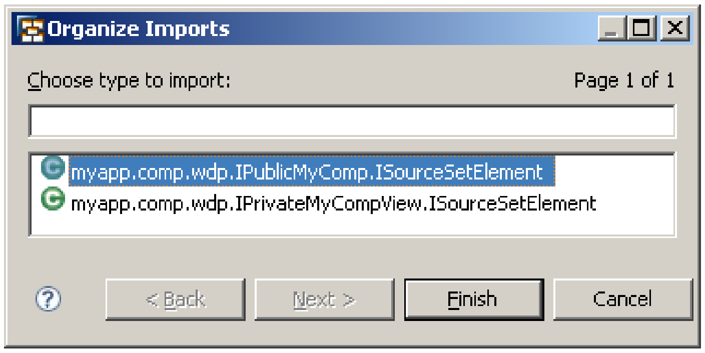
We now need to populate the *SourceSet* context node with all the possible data, and then when the user performs a filter action, we will transfer only those rows that match the filter criteria from the *SourceSet* context node to the *TableSet* context node. In this way, we can leave the Table UI element bound to the *TableSet* context node, knowing that what ever filter options the user selects, the table will always show the correct information.

7. Go to the *createStock()* method and replace references to node *TableSet* with references to node *SourceSet*. You need to change the references to ***Table****SetElement* to ***Source****SetElement*.

8. and insert the line shown in bold italic to set the IDs for the products.

```
for ( int i = 0; i < stockLen; i++ ) {
   ISourceSetElement product = wdContext.createAndAddSourceSetElement();
   product.setID(STOCK[i][0]);
   product.setAvailable((i%6 == 0) ? false : true);
   product.setArticle(STOCK[i][1]);
   product.setColor(STOCK[i][2]);
   product.setPrice( new BigDecimal(STOCK[i][3]));
   d = ( i % 2 == 0 ) ? (i+4)%30 : d;
   m = ( i % 2 == 0 ) ? i%12 : (i-1)%12;
   y = ( i % 3 == 0 ) ? 2000+i : y;
   myCalendar.set(y,m,d);
   tmpDate.setTime(myCalendar.getTimeInMillis());
   product.setDate(tmpDate.valueOf(tmpDate.toString()));
 }
```

9. *Organize Imports* (*Ctrl+Shift+o*), *choose myapp.comp.wdp.**IPublicMyComp**.ISourceSetElement*, confirm with *Finish*.


Organize Imports

Choose type to import: Page 1 of 1

myapp.comp.wdp.IPublicMyComp.ISourceSetElement
myapp.comp.wdp.IPrivateMyCompView.ISourceSetElement

10. *Save* the project.

MyCompView Controller

1. Switch to the *Java Editor* of *MyCompView*. In the *Outline View* select the *wdDoModifyView()* method and insert the highlighted line of code:

```
if (firstTime) {
  IWDTable table = (IWDTable)view.getElement("ProductsCatalogue");

  wdContext.currentContextElement().setTableSorter( new
      TableSorter( table, wdThis.wdGetSortAction(), null ) );

  wdContext.currentContextElement().setTableFilter(
      new TableFilter( table, wdThis.wdGetFilterAction(),
                            (IWDNode)wdContext.nodeSourceSet(), null ) );
}
```

2. *Organize Imports* (*Ctrl+Shift+o*) and *save* the project. Ignore the error that `wdGetFilterAction` is undefined; this will be solved in one of the next steps.

3. Double-click the *MyCompView* and switch to the *Actions* tab.

4. Press the *New…* button in the upper right section to add an action. Name the action **Init**, confirm with *Finish*, choose *save* from file menu and *rebuild* the project.

5. Jump to its implementation by selecting the *Init* method and press the *F3 key* or browse to it via context menu and add following code between the `//@@begin onActionInit(ServerEvent)` and `//@@end` tags:

```
delFilterValues();
wdThis.wdGetMyCompController().init();
```

6. Add another action and name it **Filter**. After confirming with *Finish*, *save* the project, *rebuild* it and jump to the Filter method implementation and insert following code between the `//@@begin onActionFilter(ServerEvent)` and `//@@end` tags and save:

```
wdContext.currentContextElement().getTableFilter().filter(
        wdContext.nodeSourceSet(), wdContext.nodeTableSet());
```

7. In step 5, we added a call to a view controller method called `delFilterValues()`. This method is needed to reset the table back to its initial state after a filter has been applied. At the end of the Java file of MyCompView copy following code between the `//@@begin others` and `//@@end` tags:

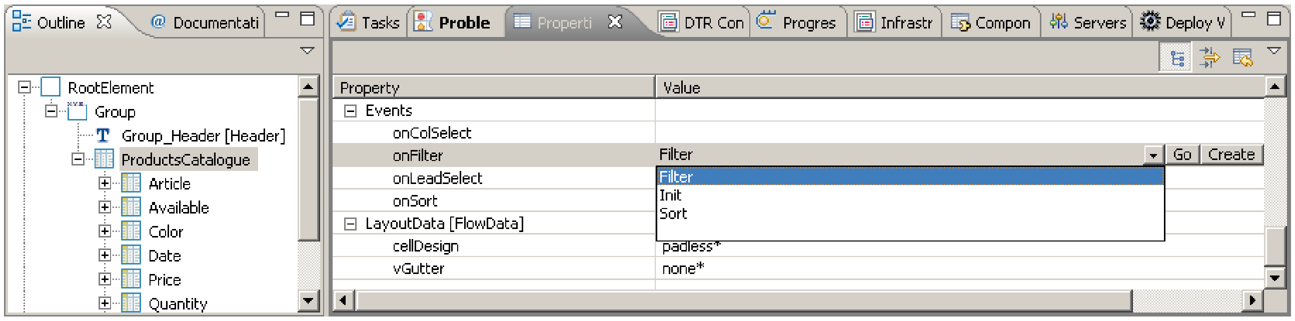
```
public void delFilterValues( )  {

  int numFilterAttr = wdContext.nodeFilterAttributes().size();
  IWDNodeInfo nodeInfo = wdContext.nodeFilterAttributes().getNodeInfo();
  IWDNodeElement element = null;

  for (int i = 0; i < numFilterAttr; i++) {
    element = wdContext.nodeFilterAttributes().getElementAt(i);

    for (Iterator <? extends IWDAttributeInfo > it =
            nodeInfo.iterateAttributes(); it.hasNext();) {

      IWDAttributeInfo attrInfo = (IWDAttributeInfo)it.next();
      element.setAttributeValue(attrInfo.getName(), "" );

    }
  }
}
```

8. *Organize Imports* (*Ctrl+Shift+o*) and choose *java.util.Iterator* and confirm with *Finish*.

Binding the filter action and values

In order to make the Table UI element aware that it has filtering capability, you must associate the *onFilter* event with the Filter action.

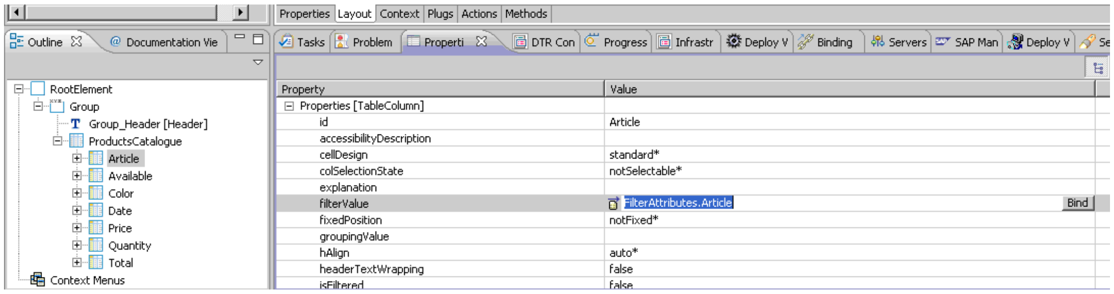
9. Double-click *MyCompView* and switch to the *Layout* tab. Then switch to the *Properties* tab in the lower screen section and select in the *Outline* view the Table element *ProductsCatalogue*. In the drop down list for the *onFilter* event, select the *Filter* action.



Property	Value
Events	
onColSelect	
onFilter	Filter
onLeadSelect	
onSort	
LayoutData [FlowData]	
cellDesign	padless*
vGutter	none*

Next, as soon as a *TableColumn* is bound to an appropriate filter action, and new first row is created automatically in which the filter values can be entered.

1. Stay in the *Layout* tab and select the *TableColumn* element *Article*. Bind its *filterValue* property to its corresponding context attribute specified in the *FitlerAttributes* node.
2. Repeat the above step for each table column.



Property	Value
Properties [TableColumn]	
id	Article
accessibilityDescription	
cellDesign	standard*
colSelectionState	notSelectable*
explanation	
filterValue	FilterAttributes.Article
fixedPosition	notFixed*
groupingValue	
hAlign	auto*
headerTextWrapping	false
isFiltered	false

3. *Save* the project*, Deploy new Archive and Run*. Filtering and sorting should now be working.

Additional functionality and handling

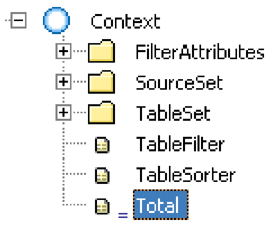
We now have a table that can be filtered and sorted, but we can´t manipulate the tables rows, nor reset the table back to the initial list.

In order to implement this functionality, we simply put a toolbar to the table UI and assign three buttons to it: *Init*, *ShowAll* and *DeleteRow(s)*.

And since the *Total_per_Article* attribute exists, we will use this to calculate the total price per row and the total of all rows together.

Calculating the totals

1. Add a new attribute of type *decimal* to the *Component Controller* context, name it **Total**, select the *calculated* option, *save* and *rebuild* the project to cause Web Dynpro to generate its *Getter* method.



Context
FilterAttributes
SourceSet
TableSet
TableFilter
TableSorter
Total

2. Switch to the *Methods* tab, select the *getTotal* Method, press the *F3 key* to jump to its implementation or use the context menu and insert following code between the `//@@begin getTotal` and `//@@end` tags:

```
BigDecimal total = new BigDecimal(0);
ITableSetNode tabSetNode = wdContext.nodeTableSet();
int n = tabSetNode.size();
for (int i=0; i<n; ++i) {
  total = total.add(
             tabSetNode.getTableSetElementAt(i).getTotal_per_Article());
}
return total;
```

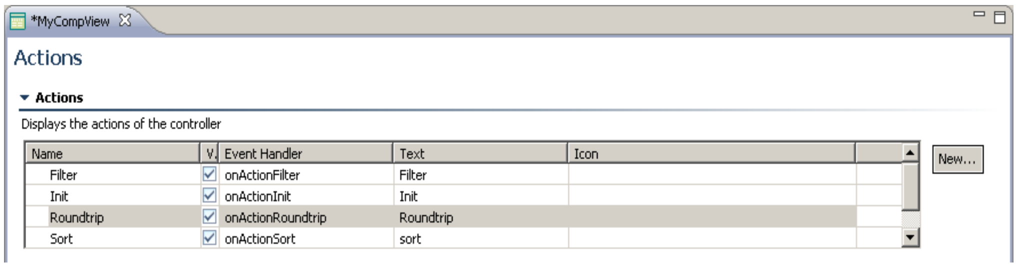
3. *Organize Imports* (*Ctrl+Shift+o*), choose *myapp.comp.wdp.**IPublicMyComp**.ITableSetNode* and confirm with *Finish*.

4. Double-click *MyComp* in the project structure and double-click then the data link between the *MyCompView* and the *Component Controller* icons. By using drag and drop map the *Total* attribute to the *MyCompView* context and confirm with *Finish*.

5. Double-click the *Component Controller* in the project structure, switch to the *Methods* tab, select the method *getTableSetTotal_per_Article()* and jump to its implementation by pressing the *F3 key* and insert the following code between the `//@@begin getTableSetTotal_per_Article` and `//@@end` tags:

```
java.math.BigDecimal sum = null;
if( element.getQuantity() < 0 )
{
  //ensure that no negative prices can be calculated
  sum = new BigDecimal(0);
}else{
  //calculate total
  sum = new BigDecimal
     (element.getQuantity()).multiply(element.getPrice());
}
  return ( sum );
```

6. Double-click *MyCompView* in the project structure and switch to the *Actions* tab, insert an action by pressing the *new…* button and name it **Roundtrip**. Confirm with *Finish*.



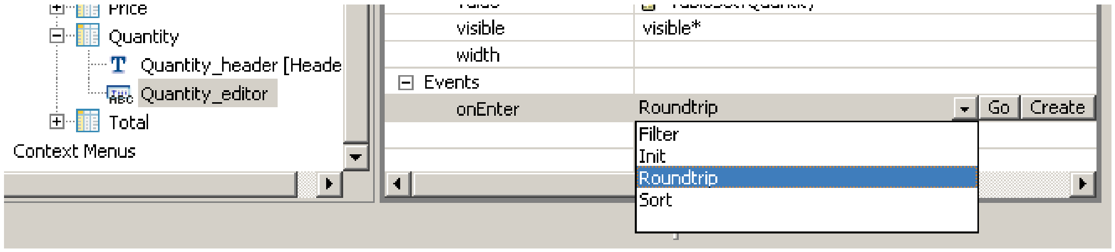
Actions

Actions

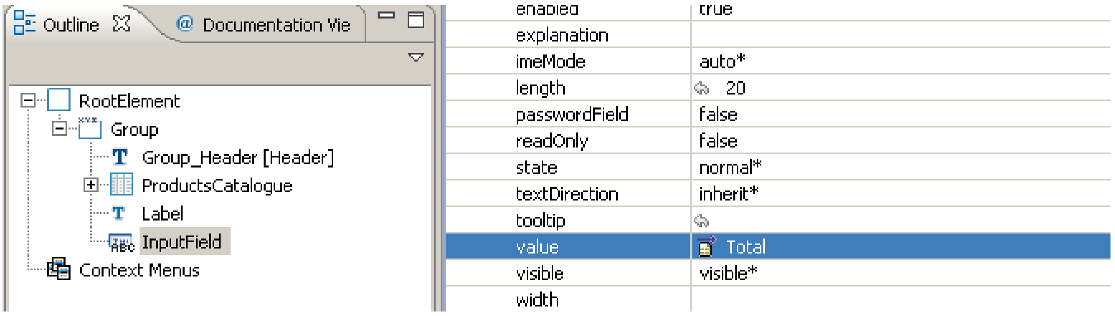
Displays the actions of the controller

Name	V.	Event Handler	Text	Icon
Filter	✓	onActionFilter	Filter	
Init	✓	onActionInit	Init	
Roundtrip	✓	onActionRoundtrip	Roundtrip	
Sort	✓	onActionSort	sort	

7. Switch to the *Layout* tab, open in the *Outline* View tree the *TableColumn* element *Quantity* and select the *Quantity_editor InputField* element. In the *Properties* tab scroll down to the *onEnter* event and choose the *Roundtrip()* action from the dropdown menu.



8. Right-click the *Group* element in the *Outline View* tree and choose *insert child…* from the context menu. Then select a *Label* UI element and click *OK*. Set its *text* property to **Total**. Now insert an *InputField* UI element and bind its *value* property to the mapped *Total* attribute.



enabled	true
explanation	
imeMode	auto*
length	20
passwordField	false
readOnly	false
state	normal*
textDirection	inherit*
tooltip	
value	Total
visible	visible*
width	

9. *Save* the project, *Deploy new Archive and run*.

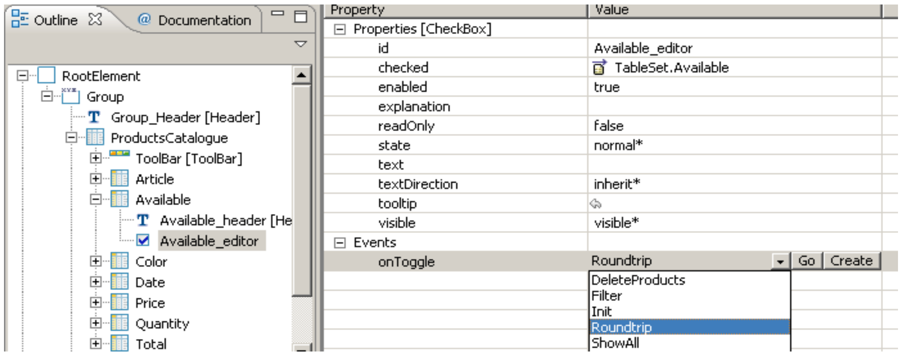
Recovering data changes

Due to the fact, that the table UI element displays data from a filtered data set, any changes made to this data won´t affect the source data set. In order to transfer the modified data from the *TableSet* context element (visualized by the *ProductsCatalogue* Table UI element) back to the source node *SourceSet*, we must use *`updateAllDataNodeElement()`* method provided by the TableFilter class.

1. Double-click *MyCompView* in the project structure, switch to the *Actions* tab, select the *Roundtrip* action and press the *F3 key* to jump to its implementation and add the following code between the `//@@begin onActionRoundtrip(ServerEvent)` and `//@@end` tags:

```
wdContext.
  currentContextElement().
    getTableFilter().
      updateAllDataNodeElement(
        wdContext.nodeSourceSet(),
          wdContext.nodeTableSet(), "ID", false);
```

2. Double-click *MyCompView* in the project structure again, Switch to the *Layout* tab, locate the *TableColumn* UI element *Available* in the *Outline View*, select the *checkbox* UI element *Available_editor* and assign the *Roundtrip* method to its *onToggle* event in the *Properties* tab.



Property	Value
Properties [CheckBox]	
id	Available_editor
checked	TableSet.Available
enabled	true
explanation	
readOnly	false
state	normal*
text	
textDirection	inherit*
tooltip	
visible	visible*
Events	
onToggle	Roundtrip

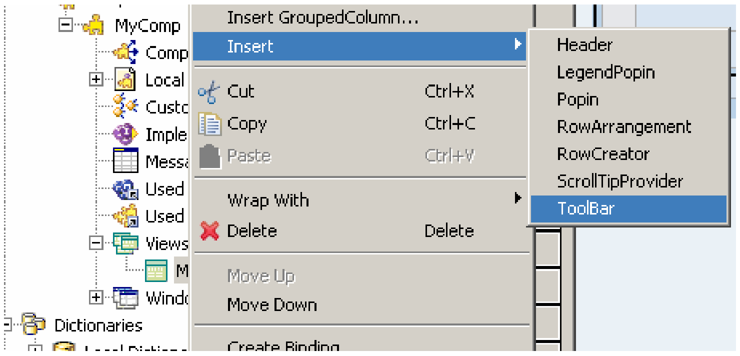
Deleting rows

1. Double-click the *MyCompView* in the project structure, switch to the *Actions* tab, add a new action, name it **DeleteProducts** and confirm with *Finish*. *Save* and *rebuild* the project, jump to the new action implementation by pressing the *F3 key* or using the context menu and insert following code between the `//@@begin onActionDeleteProducts(ServerEvent)` and `//@@end` tags:

```
ArrayList<String> myArrList = new ArrayList<String>();
ITableSetNode tabSetNode = wdContext.nodeTableSet();
int n = tabSetNode.size();
int leadSel = tabSetNode.getLeadSelection();

// loop backwards to avoid index troubles
for (int i = n-1; i >= 0; --i) {
   if(tabSetNode.isMultiSelected(i) || leadSel == i )
     myArrList.add(tabSetNode.getTableSetElementAt(i).getID());
}

wdContext.
  currentContextElement().
    getTableFilter().
      deleteAllDataNodeElement(
        wdContext.nodeSourceSet(), tabSetNode, "ID", myArrList);
```

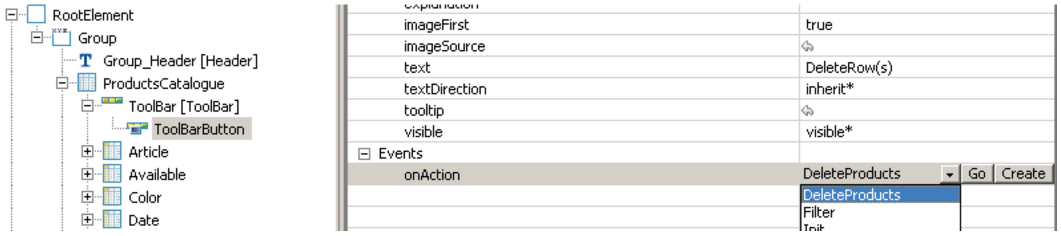
2. *Organize Imports* (*Ctrl+Shift+o*), choose *myapp.comp.wdp.IPrivate***MyCompView***.ITableSetNode*, confirm with *Finish* and *Save*.

3. Double-click the *MyCompView* in the project structure, switch to the *Layout* tab and select the table UI element *ProductsCatalogue* with right button and choose *Insert > ToolBar* from the context menu.



4. Select the *ToolBar* UI element with the right mouse button and choose *insert ToolBarItem*… In the next window select a *ToolBarButton* and confirm with *OK*.

   Set its *text* property to **DeleteRow(s)** and select the *DeleteProducts* action from the dropdown menu for its *OnAction* event.



RootElement
Group
Group_Header [Header]
ProductsCatalogue
ToolBar [ToolBar]
ToolBarButton
Article
Available
Color
Date

imageFirst	true
imageSource	
text	DeleteRow(s)
textDirection	inherit*
tooltip	
visible	visible*
Events	
onAction	DeleteProducts

   Remember, in order to be able to select multiple table rows (using *CTRL-click*), the *Selection Cardinality* property of the *TableSet* node held in the *Component Controller* context must be set to [0..n].

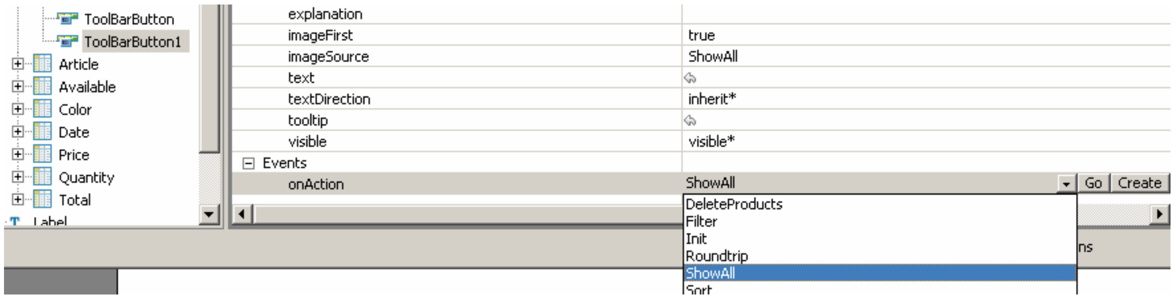
Showing the entire table data

Imagine that you filtered the catalogue for all blue colored articles and you want to reset the list back to its original state. We can put this little functionality in a button action too.

1. Double-click the *TableCompView* in the project structure, switch to the *Actions* tab, add a new action and name it **ShowAll** and confirm with *Finish*. *Save* and *rebuild* the project, jump to the action´s implementation by pressing the *F3 key* or via context menu and insert following code between the `//@@begin onActionShowAll(ServerEvent)` and `//@@end` tags:

```
// initialise filterValues
delFilterValues();
// copy sourceSet to tableSet
wdThis.wdGetMyCompController().setTableSet();
```

2. Double-click the *MyCompView* in the project structure again and switch to the *Layout* tab. Add a further *ToolBarButton* to the previously created table *ToolBar* UI element in the *Outline View*, edit its text property to **ShowAll** and assign the *ShowAll* action from the pulldown menu to its *onAction* event.



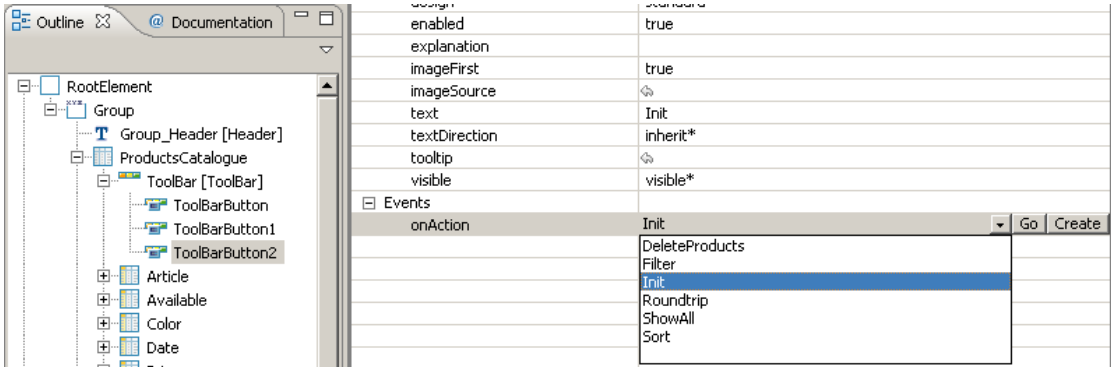
ToolBarButton
ToolBarButton1
Article
Available
Color
Date
Price
Quantity
Total

explanation	
imageFirst	true
imageSource	ShowAll
text	
textDirection	inherit*
tooltip	
visible	visible*
Events	
onAction	ShowAll

Initialize the source data

Last but not least, to provide the ability to return to the initial state of the source data, we will add a third *ToolBarButton* to the table´s *ToolBar*.

1. Double-click the *MyCompView* in the project structure, switch to the *Layout* tab and insert a new toolbar button to the table´s *ToolBar* in the *Outline View*. Set its text property in the *Properties* tab to **Init** and assign the *init* action from the pulldown menu to its *onAction* event.



- RootElement
  - Group
    - Group_Header [Header]
    - ProductsCatalogue
      - ToolBar [ToolBar]
        - ToolBarButton
        - ToolBarButton1
        - ToolBarButton2
      - Article
      - Available
      - Color
      - Date

enabled	true
explanation	
imageFirst	true
imageSource	
text	Init
textDirection	inherit*
tooltip	
visible	visible*
Events	
onAction	Init

2. *Save* the project, *Deploy new Archive and Run*.

That’s it.

Related Content

UI Element Guide: Table

Parameter Mapping

Binding Tables

Blog: Table Filter

SDN: How to Iterate over a context node

For more information, visit the User Interface Technology homepage.

Copyright