How to Implement a Web Dynpro for Java Table with Sorting and Filtering Capabilities

**Applies to:**
Web Dynpro for Java 7.11. For more information, visit the [Web Dynpro Java homepage](http://sdn.sap.com).

**Summary**
The tutorial shows how to implement sorting and filtering in a Web Dynpro table. It also offers two support classes for this purpose: TableSorter and TableFilter. It is an update of an old tutorial for release 7.0.

Filtering is based on the interface IWDMappingFilter which has been introduced in 7.10.

Further features shown in this document: adding and removing rows, a ToolBar with some buttons, calculating totals via calculated attributes, Core Component Types/Core Data Types.

**Author:** Web Dynpro Java Team  
**Company:** SAP AG  
**Created on:** 29 June 2010
# Table of Contents

Introduction ............................................................................................................................................. 3

Prerequisites ........................................................................................................................................... 4
  Systems, Installed Applications, and Authorizations ................................................................. 4
  Objectives ............................................................................................................................................ 4

First Steps ............................................................................................................................................... 5
  Creating a New Web Dynpro DC ................................................................................................. 5
  Creating the Context for the Table Data ..................................................................................... 5
  Mapping the View Context onto the Component Controller Context ........................................... 6
  Designing the View Layout ............................................................................................................. 7
  Providing Sample Data ................................................................................................................... 10

Implementing Sorting ............................................................................................................................ 13
  Importing the TableSorter Class ................................................................................................. 13
  Enhancing the Context .................................................................................................................... 14
  Remarks............................................................................................................................................. 16

Implementing Filtering ............................................................................................................................ 17
  Filtering Capabilities ...................................................................................................................... 17
  Importing the TableFilter Class ..................................................................................................... 17
  Enhancing the Context .................................................................................................................... 18

Methods and Actions .............................................................................................................................. 19
  TutorialView Controller .................................................................................................................. 19
  Binding the Filter Action and Values ............................................................................................. 19
  Remarks............................................................................................................................................. 21

Additional Functionality and Handling ................................................................................................. 22
  Calculating the Totals ....................................................................................................................... 22
  Deleting Rows ................................................................................................................................... 24
  Showing the Entire Table Data ....................................................................................................... 25
  Initializing the Source Data ............................................................................................................. 25

Further Information ............................................................................................................................... 27

Legend ..................................................................................................................................................... 27

Related Contents .................................................................................................................................... 28

Copyright ............................................................................................................................................... 29
Introduction

This tutorial focuses on the Table UI element and shows you how to use Java classes to add custom functionality in Web Dynpro. This tutorial shows you how to write custom Java classes that can react to the Table UI element’s sorting and filtering events.

First, you see how to display and edit context data using the Table UI element, which is rather easy. We implement methods to add, delete, and copy rows of the table. Lastly, the most important scenario is the sorting and filtering of table data.

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

The tutorial is an update of an old tutorial for release 7.0. It also aims to demonstrate the new, extended possibilities regarding sorting, filtering, and data types in 7.1. In addition, there is a section describing the differences to the previous tutorial.
Prerequisites

Systems, Installed Applications, and Authorizations
You need an SAP NetWeaver Developer Studio (NWDS), release 7.1 or higher, to compile and deploy the tutorial application. The application server should be of the same version as the NWDS or a higher version.
The tutorial application is available as a development component (DC). You need to import the software component HM-WDUIDMKTCNT, which contains the DC tc/wd/tut/table/sofi. The exact steps are described in a separate document.

Objectives
In this tutorial, you learn how to:
- Create the context of the component controller and its mapping with the view controller
- Create a Table UI element and bind 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
First Steps

Creating a New Web Dynpro DC

Create a new Web Dynpro DC called `tc/wd/tut/table/sofi` containing a Web Dynpro application `TutorialApp` and a Web Dynpro component `Tutorial`. Leave the `Default Window and Views` option selected in order to them applied by default. Then rebuild the project so that the various Java source files 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 `DataSet`. The `collection` cardinality of the node should be left at the default value `0...n`.

2. Change the `selection` cardinality to `0...n`. Both properties can be found in the `Properties` view for the controller.

3. Add attributes to this node and set the data types according to the table below. `Amount` and `Date` are Core `Component Types`. You can choose `Browse…` in the `Type Selection` dialog to select them.

4. For the `Total_per_Article` attribute, set the `calculated` property to `true`. This causes the NWDS to generate the appropriate `accessor` (get/set) methods.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type</th>
<th>Calculated?</th>
<th>Read-only?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article</td>
<td>string</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Available</td>
<td>Boolean</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Color</td>
<td>string</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Date</td>
<td>com.sap.dictionary.ccts.Date</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Price</td>
<td>com.sap.dictionary.ccts.Amount</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Quantity</td>
<td>integer</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>Total_per_Article</td>
<td>com.sap.dictionary.ccts.Amount</td>
<td>yes</td>
<td>true</td>
</tr>
</tbody>
</table>

The context of the `Component Controller` should now look as follows:
Mapping the View Context onto the Component Controller Context

In order to make the data stored in the context of the component controller available to the view controller, we need to tell the view controller (called TutorialView) 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 icon for the Tutorial component to open the graphical controller editor. Create a data link from the TutorialView to the component controller. This causes the component controller to be added to the view controller’s list of required controllers.

2. Use drag and drop, drag the DataSet node from the component controller context on the right (Tutorial) to the left and drop it on the view controller’s context root node (TutorialView). Deselect and select again the DataSet node checkbox to include all attributes in the mapping. Confirm by choosing Finish.

The mapping wizard has now created a node called DataSet in the context of the view controller and mapped all its attributes to the DataSet node (the mapping origin node) in the component controller’s context.

Any code that accesses data in the DataSet node of the view controller actually accesses the data in the context of the component controller.
**Designing the View Layout**

After the view controller context has been created, we can start to add UI elements to the view layout in order to display the context data on the screen.

To present all the articles on the browser screen, we just have to add some UI elements to the view layout.

1. Open the *TutorialView* editor by double-clicking the icon or using the context menu. Switch to the *Layout* tab.
2. In the *Outline view*, delete the *DefaultTextView* element if necessary.
3. Right-click the *Root Element* of the view and choose *Apply Template*.
   
   Select the *Table* icon, overwrite the *Template Instance Name* with the value *ProductsCatalogue*, and choose *Next*. (You can use any valid identifier here but it is important to use the same identifier later in the code that accesses the table at runtime.)
4. Select the *DataSet* context node. By default, all attributes are selected automatically, so just choose *Next*:

![Select Attributes](image)

5. Change the editor for the *Quantity* attribute from *TextView* to *InputField* and confirm by choosing *Finish*.

![Specify Table Properties](image)

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
The following screenshots show the complete property settings for the table and how the UI appears in the preview area:
Providing Sample Data

In order for the table to display some data, we implement a supply function for the DataSet context node.

1. Double-click the Component Controller
2. Switch to the Context tab and select the DataSet node.
3. Set the node’s property Supply Function. Click inside the value cell and choose the button with the ellipsis (three dots). In the dialog box, confirm the proposed name, supplyDataSet.

4. Choose Ctrl+S to save the changed controller and rerun the code generator. This recreates the Java code for the controller, including the new supply function we have just declared.
5. Navigate to the new supplyDataSet() method by choosing Go.
6. Between the //@@begin and //@@end tags, insert the following code:

   ```java
   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;
   CctCode eur = new CctCode("EUR", null, null, null, null, null);
   for ( int i = 0; i < stockLen; i++ ) {
      IDataSetElement product = node.createAndAddDataSetElement();
      product.setAvailable((i%6 == 0) ? false : true);
      product.setArticle(STOCK[i][0]);
      product.setColor(STOCK[i][1]);
      product.setPrice(new CctAmount(new BigDecimal(STOCK[i][2]), eur));
      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());
Date date = new Date(myCalendar.getTime().getTime());
product.setDate(new CctDate(date));
}

7. There is no need to write any code that calls the supply method explicitly. Instead it is called by the framework automatically whenever the node is empty or invalid and its data is requested, for example when a Table UI element is bound to that node.

8. Organize Imports (Ctrl+Shift+O)
Choose com.sap.test.tc.wd.tut.table.sofi.wdp.IPublicTutorial.IDatasetElement and confirm by choosing Finish.

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

```java
private static final String STOCK[][] = {
    {"Jacket","blue","34.60"},
    {"Skirt","red","24.95"},
    {"T-Shirt","orange","29.90"},
    {"Trousers","black","64.90"},
    {"Top","black","44.90"},
    {"Dress","colored","78.90"},
    {"Blouse","white","35.50"},
    {"Jeans","blue","89.90"},
    {"Pullover","red","69.00"},
    {"Sweatshirt","green","61.60"},
    {"Polo Shirt","yellow","14.65"},
    {"Shorts","dark blue","44.90"},
    {"Blouse","white","49.90"},
    {"Skirt","white","55.00"},
    {"Pullover","white","127.00"},
    {"Dress","black","178.90"},
    {"Top","red","54.00"},
    {"Trousers","black","79.00"},
    {"T-Shirt","red","45.60"},
    {"Jacket","white","55.80"},
    {"Pullover","white","130.90"},
    {"Jacket","blue","200.90"},
    {"Skirt","brown","89.90"},
    {"Alpaca Pullover","brown","230.00"},
    {"Lambswool Pullover","yellow","130.00"}
};
```

10. Save the project and select Deploy New Archive and Run from the context menu of the application.
The following screen should be displayed in your browser:
Implementing Sorting

In the releases SAP NetWeaver 2004 and SAP NetWeaver 7.0, Web Dynpro did not provide any sorting logic by default. This had to be implemented explicitly by the application developer. The SAP NetWeaver 7.0 version of this tutorial offers a separate Java class (called TableSorter) to help the application developer with this task. Although the NW 7.1 version of Web Dynpro is now able to sort a node directly, this tutorial continues to use an updated version of the TableSorter class that supports the new 7.1 features.

Web Dynpro can sort a node via the sortElements(ICMISortCriterion) method. This method is even able to sort by multiple columns.

Importing the TableSorter Class

1. Select Import from the File menu.
2. Choose General → File System and choose Next.
3. Specify the folder where the Java classes are stored (TableSorter.java and its base class TableHelper.java) and select them by selecting the corresponding checkbox. You may also import TableFilter.java, we will need it later.
4. Now specify where to store the class in the current project structure by using the second browse button Into Folder. Navigate through the project structure as follows: Resources → src → packages → com → sap → test → tc → wd → tut → table → sofi → util and confirm by choosing OK.
5. Manually append a new folder to this path and name it util, so that the path specified describes the location src\packages\com\sap\test\tc\wd\tut\table\sofi\util.

Confirm by choosing Finish.

If you choose a different location, you need to change the package for the Java class accordingly.

Import Window:

<table>
<thead>
<tr>
<th>Import Window:</th>
<th>Resulting Project Structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Import Window" /></td>
<td><img src="image" alt="Resulting Project Structure" /></td>
</tr>
</tbody>
</table>
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 a table column header, we need to trigger the table’s onSort event to sort the table correctly.

1. Double-click the Component Controller and switch to the Context tab.

2. Add an attribute named TableSorter to the context root node 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 is found, select the one matching your project’s structure path, for example util, choose OK and confirm by choosing Finish.

3. Double-click the Tutorial component in the project structure, double-click the data link between the TutorialView and Component Controller icons and map the TableSorter attribute to TutorialView.

4. Double-click TutorialView and add a new action. Switch to the Action tab and choose the New… button on the upper right-hand side. Name the action Sort and choose Finish.
5. Open its method implementation by pressing the F3 key or via the context menu. Add the following code between the //@@begin and //@@end tags:

```java
wdContext.currentContextElement().
    getTableSorter().sort(wdEvent, wdContext.nodeDataSet());
```

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

```java
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 TutorialView and select the `ProductsCatalogue` table UI element from the Outline view in the bottom left pane. Select the Properties tab and associate the table’s `onSort` event with the Sort action we just created.

9. Save, deploy the new archive, and run it.

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

The TableSorter class has additional capabilities not shown in this tutorial:

- You can specify which columns can be sorted, instead of letting the class automatically choose all columns that can be sorted.
- You can specify your own comparator for any column if the default sorting algorithm does not match your needs. (IWDNode.setSortCriterion does not offer this.)
- TableSorter can sort entries even if a column uses an attribute of a child of the table’s source node. (IWDNode.setSortCriterion cannot do this.)

For more information, see the Javadoc.

The major differences between TableSorter in this tutorial and the one from the SAP NetWeaver 7.0 tutorial are as follows:

- All string comparisons can be done using a collator instead of simple string comparisons. This allows locale-specific sorting. For compatibility reasons between versions (for example after an upgrade), this is switched off by default.
- Core component types are supported. They are compared by means of the content component.
- You can create a matching comparator for a given data type as a basis for your own comparator.
- You can exchange the comparator for a given column at any time. Any subsequent sorts then use the new comparator.
Implementing Filtering

Filtering is a tedious topic in Web Dynpro in SAP NetWeaver 7.0. In order to apply filtering to a table, we previously needed a total of three context nodes:

- One for the table data (which we already have)
- One to hold the filter values
- One to hold the subset of rows after filtering has been applied

All three nodes needed to have the same structure and had to be located directly under the root node (I.E. Independent nodes). Additionally, we needed an attribute that uniquely identifies each element.

With Web Dynpro 7.1 however, this has become much easier. Now Web Dynpro can apply filtering at the level of context mapping. The application can decide for each element of the data node whether it is visible in the mapped node or not. This can be achieved by attaching an IWDMappingFilter to the mapped node’s IWDNodeInfo.

Nonetheless, there is still some work left to do for the application developer. For this reason, this tutorial provides a pre-written class, called TableFilter, which makes filtering really easy.

Filtering Capabilities

The table filter has the following capabilities:

- **Strings:**
  If the filter value contains simply the letter “a”, this is treated as if you had entered “*a*” and returns any field that contains an “a” at 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 searches either for the exact value, or if the following operators are used, ranges, inclusions, and exclusions can be specified.
  If you use the inclusion (“=”) or exclusion (“#”) operators, then these should always appear first.

  **For ranges:**
  “~100” specifies all values up to and including 100
  “1~100” specifies all values between 1 and 100 inclusive
  “100~” specifies all values greater than or equal to 100

- **Boolean values:**
  Booleans can be filtered using the inclusion (“=”) or the exclusion (“#”) operators in the first position.

Importing the TableFilter Class

If you did not import the TableFilter together with TableSorter and TableHelper, do it now. Proceed as described in the Implementing Sorting section.
Enhancing the Context

We have to create and save a TableFilter just like the TableSorter and we need to have various context attributes available that can store the filter values. Since these attributes are only relevant for filtering, they will reside in the view controller’s context rather than being mapped from the component controller.

1. Double-click the View Controller and switch to the context tab.
2. 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 as you did for the TableSorter attribute.
3. Create a new node under the context root node called FilterAttributes. Set its collection cardinality property to 1..1. Additionally, the initializeLeadSelection property should be set.
4. Add the attributes Article, Available, Color, Date, Price, Quantity, and Total_per_Article, all of type string, not calculated and not read-only. These attributes correspond to the table columns that can be used for filtering.

The context should now look as follows:
Methods and Actions

TutorialView Controller

1. Double-click the TutorialView and switch to the Actions tab.
2. Add another action and name it Filter.
3. Switch to the Java Editor of TutorialView. In the Outline View, select the wdDoModifyView() method and insert the highlighted line of code:

   ```java
   if (firstTime) {
     IWDTTable table = (IWDTTable)view.getElement("ProductsCatalogue");
     wdContext.currentContextElement().setTableSorter(
       new TableSorter(table, wdThis.wdGetSortAction(), null));
     wdContext.currentContextElement().setTableFilter(
       new TableFilter(table,
          wdThis.wdGetFilterAction(),
          wdContext.nodeDataSet(), null));
   }
   ```

4. Organize Imports (Ctrl+Shift+O), save the project, and then rebuild it.
5. Jump to the Filter method implementation, insert the following code between the //@@begin and //@@end tags, and save it:

   ```java
   wdContext.currentContextElement().getTableFilter().filter();
   ```

Binding the Filter Action and Values

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

1. Double-click TutorialView 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 dropdown list for the onFilter event, select the Filter action.

   ![Properties Tab Screenshot](image)

   As soon as a TableColumn is bound to an appropriate filter action, a new first row is created automatically in which you can enter the filter values.

2. Select the TableColumn named Article. Bind its filterValue property to the corresponding context attribute in the FilterAttributes node.
Repeat the above step for each table column.

3. Save the project, deploy the new archive and run it.
   Filtering and sorting should now be working.
Remarks

Attaching a TableFilter changes the behavior of the table’s data source node substantially (due to the fact that it attaches an IWDMappingFilter), even if no filter applies yet.

Usually adding, removing, or changing the order of elements in the view controller context node directly manipulates the origin node as well and the index of an element in the mapped view node and its origin element are always identical.

**IMPORTANT!** This relation is lost when an IWDMappingFilter is attached!

Now the indexes cannot be identical anymore and as a consequence, the node stops any attempt to keep them in sync. Any changes to the order in the mapped node (for example sorting) do not affect the origin node at all, and vice versa. Additionally, the mapped node prevents any modification of the list (that is, adding or removing elements); you have to delegate this from the view to the component controller.

The TableFilter class has additional capabilities not shown in this tutorial:

- You can specify which columns can have a filter, instead of letting the class choose all columns that it can handle.
- You can specify your own matcher for any column if the default filter does not match your needs. This filter can use the attribute value and make object comparisons or it can use the displayed text and pattern matching.
- TableFilter can even filter based on a column that uses an attribute of a child of the table’s data source node.
- You do not have to supply a filter attribute yourself. TableFilter creates one automatically if none exists. However in this case, your application does not have access to the filter value.

For more information, see the Javadoc.

When upgrading an SAP NetWeaver 7.0 application to this new version of the TableFilter, you have to make substantial changes to your code.

Below, you can find a short description of this based on the SAP NetWeaver 7.0 tutorial.

In SAP NetWeaver 7.0, there were two nodes for the table data; SourceNode always contained the complete list and TableNode contained a subset of this list with only those elements that matched the filter criteria. This forced us to write quite a bit of code to keep the TableNode up-to-date.

Additionally, any modification of the TableNode had to be copied to the corresponding SourceNode. This forced us to create a unique identification of the elements, so that we could find the corresponding element in SourceNode.

In the 7.1 version, the DataNode from the component controller corresponds to the previous SourceNode and the DataNode in the view corresponds to the previous TableNode. Thanks to IWDMappingFilter filtering this is now possible without relinquishing the mapping relation. This means that attribute values (and changes) are still propagated automatically. In addition, the addition and removal of elements in the origin node are still observed in the mapped node.

Some of these issues are shown in the following section.
**Additional Functionality and Handling**

We now have a table that can be filtered and sorted, but we cannot edit the table rows yet, 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 add three buttons: `Init`, `ShowAll`, and `DeleteRow(s)`.

The `Total_per_Article` attribute is used to calculate the total price of each row, the `Total` attribute gives the total across all rows.

**Calculating the Totals**

1. Each element has a calculated attribute, `Total_per_Article`. The corresponding getter method however is still empty. We implement it by simply multiplying the price by the quantity. The coding however, gets slightly more complicated because the price is not simply a number, but a core data type.

   Insert the following code in `Tutorial.java` between the `//@@begin` and `//@@end` tags:

   ```java
   CctAmount price = element.getPrice();
   BigDecimal amount = price.getContent().multiply(
       new BigDecimal(element.getQuantity()));
   return new CctAmount(amount, price.getCurrencyCode());
   ```

2. Add a new attribute of the core data type `Amount` to the `View Controller` context, name it `Total`, select the `calculated` option, save, and rebuild the project to cause the NetWeaver Developer Studio to generate the `getTotal` method.

   ![Context](image)

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

   ```java
   IDataSetNode dataNode = wdContext.nodeDataSet();
   if (dataNode.size() == 0) {
       return null;
   }
   CctAmount totalPerArticle =
       dataNode.getDataSetElementAt(0).getTotal_per_article();
   BigDecimal sum = totalPerArticle.getContent();
   CctCode currency = totalPerArticle.getCurrencyCode();
   for (int i = 1; i < dataNode.size(); i++) {
       totalPerArticle = dataNode.getDataSetElementAt(i).getTotal_per_article();
       sum = sum.add(totalPerArticle.getContent());
   }
   return new CctAmount(sum, currency);
   ```


   ![Import](image)
5. Double-click *TutorialView* in the project structure and switch to the *Actions* tab, insert an action by choosing *New…* and name it *Roundtrip*. Confirm by choosing *Finish.*

6. Switch to the *Layout* tab and in the *Outline* view, open the *TableColumn* named *Quantity*. Select the *Quantity_editor InputField* element and on the *Properties* tab, scroll down to the *onEnter* event and choose the *Roundtrip* action from the dropdown list.

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

8. Save the project, deploy the new archive, and run it.
Deleting Rows

Theoretically, it is easy to remove all selected rows of a table. The table maintains its selection in its source node. A row is selected exactly if the source node’s multi-selection contains the corresponding node element. So removing all selected elements basically means calling `sourceNode.removeSelectedElements()`.

But if you try this here, you will notice that it does not work. You will get a `ContextException` telling that you must not modify a filtering node and that you should modify the origin instead. That is what we will do now:

1. Double-click the `Tutorial` component controller in the project structure, switch to the `Methods` tab, add a new method, name it `removeSelectedElements`, and confirm by choosing `Finish`.

   Save the project, go to the method, and insert the following code between the `//@@begin` and `//@@end` tags:

   ```
   wdContext.nodeDataSet().removeSelectedElements();
   ```

2. Double-click the `TutorialView` in the project structure, switch to the `Actions` tab, add a new action, name it `DeleteProducts`, and confirm by choosing `Finish`.

3. Save and rebuild the project, jump to the new action implementation by pressing the F3 key or using the context menu, and insert the following code between the `//@@begin` and `//@@end` tags:

   ```
   wdThis wdGetTutorialController().removeSelectedElements();
   ```

4. Double-click the `TutorialView` in the project structure, switch to the `Layout` tab and select the Table UI element `ProductsCatalogue` with the right mouse button and choose `Insert → ToolBar` from the context menu.

5. Select the `ToolBar` UI element with the right mouse button and choose `Insert ToolBarItem`... In the next window, select a `ToolBarButton` and confirm by choosing `OK`.

   Set its `text` property to `DeleteRow(s)` and select the `DeleteProducts` action from the dropdown list for its `onAction` event.

Remember, in order to be able to select multiple table rows (using CTRL+click), the `Selection Cardinality` property of the `DataSet` node in the `Component Controller` context must be set to 0..n.
Showing the Entire Table Data

Imagine that you have filtered the product catalog for all articles colored blue and you want to reset the list back to its original state. We can implement this functionality via a button action too.

It is quite simple – we just have to clear all filter attributes and trigger filtering again. Since we have put all the filter values into a single node that contains nothing else, deleting all filter values is trivial: Just invalidate the node.

1. Double-click the TutorialView in the project structure, switch to the Actions tab, add a new action and name it ShowAll, and confirm by choosing Finish.

Save and rebuild the project, jump to the action’s implementation by pressing the F3 key or via the context menu and insert the following code between the //@@begin and //@@end tags:

```java
// delete all filter attributes
wdContext.nodeFilterAttributes().invalidate();
// filter again
wdContext.currentContextElement().getTableFilter().filter();
```

This would be a good time for some code refactoring. We have written this lengthy code to call the filter twice, once in onActionFilter and once here. It is time for a new method, filter(), but this has been left as an exercise for the reader.

2. Double-click the TutorialView in the project structure again and switch to the Layout tab. Add another ToolBarButton to the existing toolbar.

3. Set its text property to ShowAll and assign the ShowAll action to its onAction event.

Initializing the Source Data

Last but not least, in order to provide the ability to return to the initial state of the source data, we will add a third button to the table’s toolbar. This task is also easy because we are using a supply function to populate the DataSet node.

In order to force the execution of a supply function, the only requirement is to invalidate the associated context node. Then, when the next attempt is made to access the contents of the node (for example, when the Table UI element is being rendered), the supply function is called automatically.

1. Double-click the Component Controller in the project structure, select the Methods tab, and add an init method with a return type of void and no parameters.

Save the project, jump to the implementation, and add the following code between the //@@begin and //@@end tags:

```java
wdContext.nodeDataSet().invalidate();
```
2. Double-click the TutorialView in the project structure, switch to the Layout tab, and insert a new toolbar button in the table's ToolBar in the Outline View. Set its text property on the Properties tab to Init and define its onAction event. Since it does not exist yet, choose Create, name the action Init and choose Finish.

3. Save the project, choose Go to jump to the implementation, and insert the following code between the //@@begin and //@@end tags:

```java
// delete all filter attributes
wdContext.nodeFilterAttributes().invalidate();
wdThis.wdGetTutorialController().init();
```

Once again we have the possibility of improving the efficiency of the coding here. Now we could factor out a deleteFilterAttributes() method.

1. Save the project, deploy the new archive, and run it.
Further Information

UI Element Guide: Table
Parameter Mapping
Binding Tables
Blog: Table Filter
SDN: How to Iterate Over a Context Node

Legend

Text Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Note]</td>
<td>Note</td>
</tr>
<tr>
<td>![Recommendation]</td>
<td>Recommendation</td>
</tr>
<tr>
<td>![Warning]</td>
<td>Warning</td>
</tr>
<tr>
<td>![See also]</td>
<td>See also</td>
</tr>
</tbody>
</table>

For more information, visit the [Web Dynpro Java homepage](https://www.sap.com).
For more information, visit the [Web Dynpro Java homepage](https://www.sap.com).
For more information, visit the [Web Dynpro Java homepage](https://www.sap.com).
Related Contents
For more information, visit the Web Dynpro Java homepage.
How to Implement a Web Dynpro for Java Table with Sorting and Filtering Capabilities

Copyright

© Copyright 2010 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

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

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

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

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

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.