Implementing Sorting on a Table Depending on Region-Specific Number Formats

Applies to:

Web Dynpro for Java applications for SAP Netweaver 04 SP Stack 17, SAP Netweaver 04s SP Stack 15. For more information, visit the User Interface Technology Homepage.

Summary

This article demonstrates the implementation of sorting functionality on the table UI element with enhanced sorting capabilities that sorts dates, strings, numeric values as well as numeric values stored as strings depending on the region-specific number format.

Author: Divyata Dal

Company: Larsen and Toubro Infotech Limited

Created on: 23 September 2009

Author Bio

Divyata Dal is working at Larsen and Toubro Infotech Limited as a SAP ABAP and SAP Netweaver Developer for Web Dynpro in Java. She has knowledge in diverse aspects of ABAP, Web Dynpro for Java, SAP Enterprise Portal, Java/J2EE and Adobe Flex.

Table of Contents

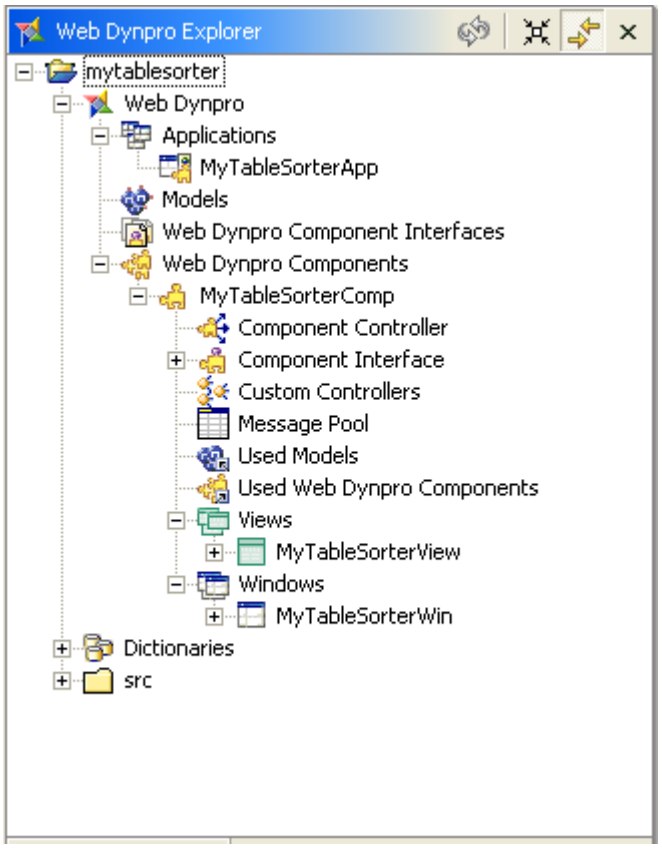
Introduction

Sorting is one of the features that can be implemented for the Table UI element. This is usually done using a custom Java class. The custom file is imported into the Web Dynpro project and the features of sorting, therefore, incorporated for the table.

The conventional sorting file facilitates sorting strings, dates and numbers. However, we will modify the sorting file to sort numeric data stored as strings specific to the format selected by the user from the portal end. How this can be achieved and what the user formats are will be explained at length in this article.

Creating a Web Dynpro Application

Create a new Web Dynpro DC named **mytablesorter** with the component named MyTableSorterComp and the application named MyTableSorterApp. Let the window be named MyTableSorterWin and its view MyTableSorterView.



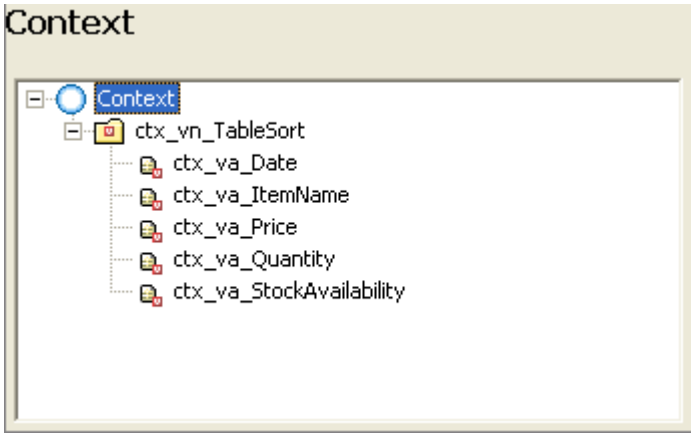
Creating a Context for the Table Data

In order to manipulate the data in the table, we will create a value node in the *Component Controller*.

1. Create a value node called 'ctx_vn_TableSort'. The cardinality of this node would be default value of [0…n]. Change the selection cardinality to [0…n].
2. Now manually add attributes to this node and set their data types according to the table given below:

Attribute Name	Data Type
ctx_va_ItemName	string
ctx_va_StockAvailability	boolean
ctx_va_Date	date
ctx_va_Quantity	string
ctx_va_Price	string

3. The 'ctx_vn_TableSort' node of the Component Controller should now look like this:

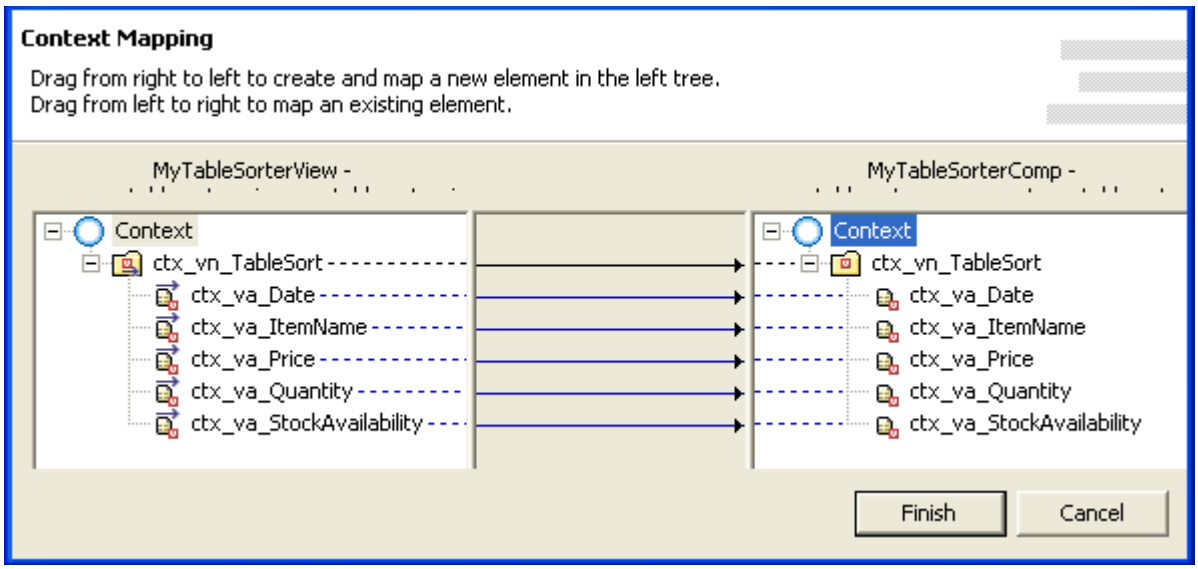


Mapping the View Context to the Component Context

The context node is created in the component controller and cannot be used directly in the view. Therefore, mapping must be done to make the node in the component controller accessible in the view. Mapping can be achieved by the following steps:

1. Double-click on the component controller.
2. Create a data link between MyTableSorterComp and MyTableSorterView.
3. Once the data link is created, double-click on this link and drag the context from the context of the component controller on the right to the context of the view on the left.
4. Select all the attributes of the node and click on 'Finish'.

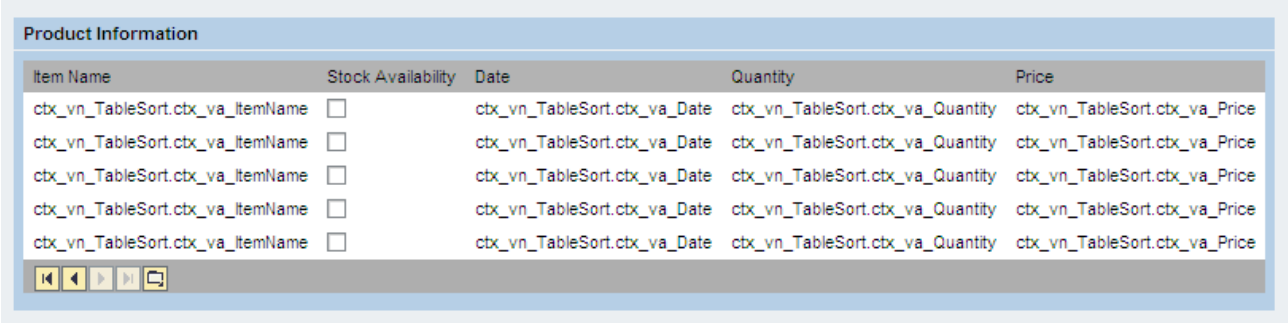
The context node ‘ctx_vn_TableSort’ has now been duplicated in the view. Now, data can be read from this node form the view as if it were being read directly from the component controller.



Designing the View

In order to display information on the screen and demonstrate sorting, we shall now proceed with designing the view.

1. Switch to the Layout tab of the view and delete the default TextView element.
2. Right click on the Root Element and add a Group it, grp_Main and set the header to **Product Information**.
3. Now right-click on the group and add a Table UI element to the group, tbl_TableProducts. This table must be mapped to a data source. For this, right-click on the table and select *Create Template*.
4. Select the ‘ctx_vn_TableSort’ node and click *Next*.
5. The table properties window appears. All the values appear as TextView input elements. Change the value of ‘Editor’ column for *Stocks Availability* to checkbox.
6. Confirm with *Finish*.
7. Change the headers of the individual columns to *Item Name*, *Stocks Availability*, *Date*, *Price* and *Quantity* respectively.



Populating Data in the Table

In order to demonstrate the data in the table, we first have to populate the table with some dummy data.

1. Switch to the *Methods* tab of the view and add a new method mPopulateTable to it.
2. Navigate to the *Implementation* section and insert the following code in the method just created:

```
String products[][] = {
             {"shirt", "true", "2009-06-19", "100", "650.00"},
             {"trouser", "false", "2009-08-01", "90", "999.90"},
             {"t-shirt", "true", "2009-12-07", "200", "349.99"},
             {"skirt", "false", "2009-05-05", "0", "550.00"},
             {"jacket", "true", "2009-10-30", "30", "990.00"},
             {"jeans", "true", "2009-10-01", "55", "1050.00"},
             {"dress", "false", "2009-17-11", "17", "1349.99"},
             {"scarf", "true", "2009-03-02", "80", "250.00"},
             {"sweater", "false", "2009-02-05", "9", "2550.00"},
             {"short", "true", "2009-10-09", "130", "650.00"}
            };
```

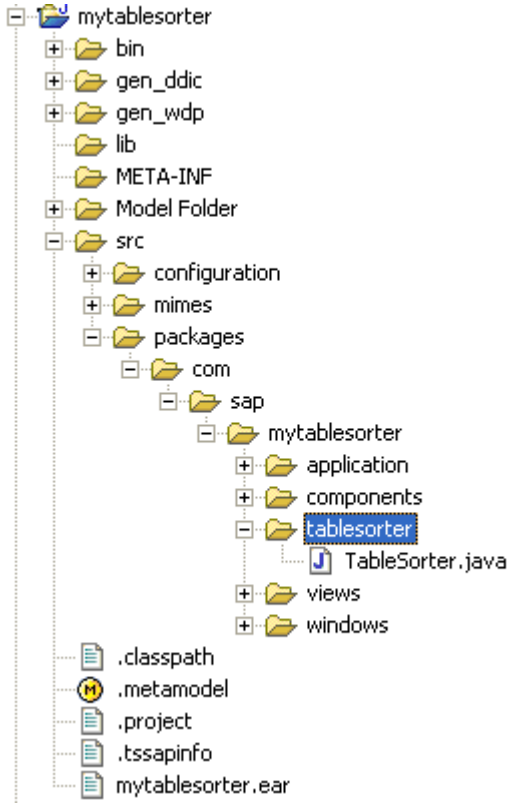
3. This method must be called in the *wdDoInit()* method of the view.

Implementing Sorting on the Table

The sorting logic on the table can be implemented by importing the Java file into the Web Dynpro project. We have the *TableSorter.java* file. This file has been prewritten to implement sorting on the table UI element. The sorting logic in this prewritten file has, however, been enhanced to work on numeric strings. The sorting file can be imported into the project using the following steps:

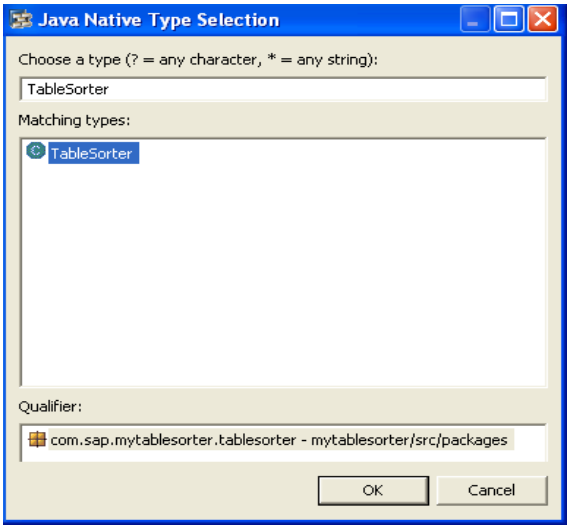
1. Select the *Navigator* tab and navigate to the folder : *mytablesorter -> src -> packages -> com -> sap -> mytablesorter*.
2. Right click on the *mytablesorter* folder select *New -> Other -> Simple -> Folder -> Next*.
3. Enter the name of the folder as *tablesorter* and confirm with *Finish*.
4. Now, the path where the *TableSorter.java* file would be stored looks like *mytablesorter -> src -> packages -> com -> sap -> mytablesorter -> tablesorter.*

5. Paste the *TableSorter.java* file at the above path.



Now, in order to access the instance of the *TableSorter* file, we must create a context attribute and map it to this class. This can be achieved with the following steps:

1. Switch to the context tab of the Component Controller
2. Create a new context attribute called ‘ctx_va_TableSorter’ and click on browse for its Java Native type. Type *TableSorter* in the input field and select the entry that appears in the lower area matching the project structure path
3. Click on OK and confirm with Finish
4. Double click on the link between the Component Controller and the View and map the context attribute ‘ctx_va_TableSorter’ from the component to the view.



5. Now create a new *Action* in the view called *aSort* and click *Finish.*
6. In the implementation of this method, add the following code:

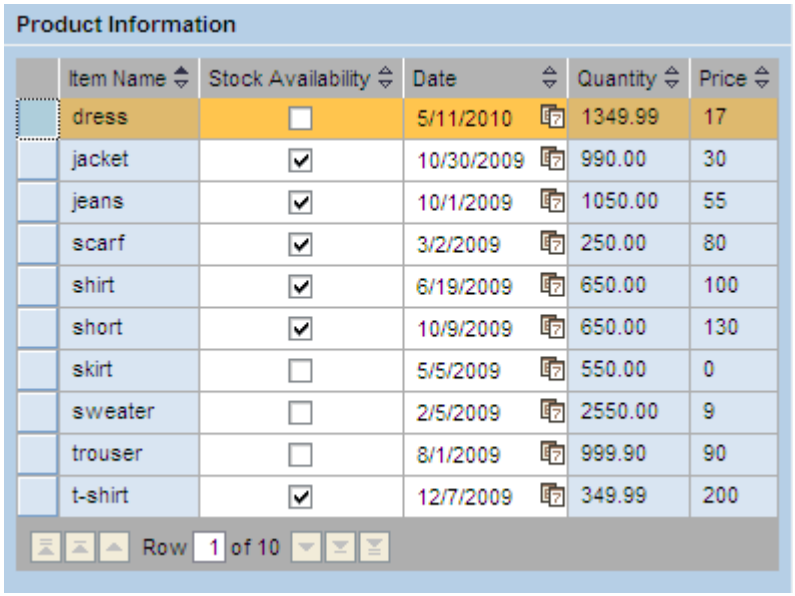
   wdContext.currentContextElement().getCtx_va_TableSorter().sort(wdEvent, wdContext.nodeCtx_va_TableSort(), l_dataFormat);

   Here *l_dataFormat* is the format selected by the user, say '--,---,---~.—'
7. Insert the following piece of code in the *wdDoModifyView()* method:

```
If(firstTime)
{
      IWDTable l_table = (IWDTable) view.getElement("tbl_TableProducts");
      wdContext.currentContextElement().setCtx_va_TableSorter(new
TableSorter(l_table, wdThis.wdGetSortAction, null));
}
```

8. Now switch to the Layout tab of the view. In the Properties tab, associate the table's *onSort* event with the *aSort* action just created
9. Save, Deploy New Archive and Run

   Now when you click on the column header, a sort icon appears on the column. Sorting in the ascending or descending order can be achieved by clicking on the column header.

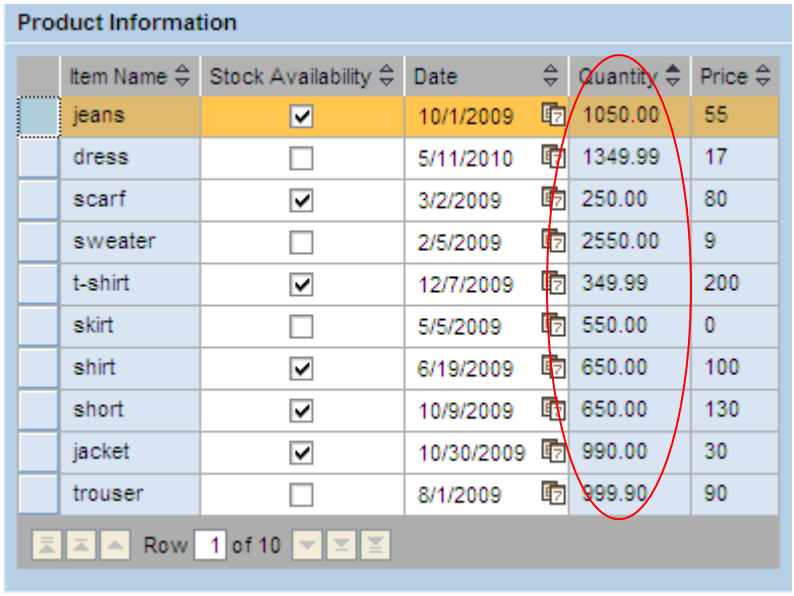


   Here, the data is sorted in ascending order of the *Item Name*.

Understanding Modifications in the TableSorter.java File

The Java file used sorts the numeric data only if they are stored as short, long integer, float or double in the context. However, in cases where numeric data is stored as string, the sorting is incorrect, since the column data is treated as strings and sorted accordingly. To overcome this shortcoming, certain modifications have been made in the Java file used for sorting so that the strings are parsed and categorized into either strings or strings holding numeric data.

For example, the *Quantity* field is stored as string with the values in floating point. When we sort the table data in the ascending order of the Quantity, we notice that the values are not sorting correctly, since they are treated as strings and not numeric data.



Formats Used for the Numeric Strings

The central idea revolves around sorting the numeric strings based on region-specific number formats. The format is taken into account to identify the format of the numeric string and then sort them accordingly. There are 3 user-formats that have been handled in the Java sorting file:

1. US Format (--,---,---~.-- / --,---,---~.---) : This is the standard format, where the grouping character is a comma and the decimal place is indicated by a period.
2. German Format (--.---.---~,-- / --.---.---~,---) : This is like the COMMA format, but it interchanges the role of the decimal point and the grouping character. Here, the grouping character is a period and the decimal place is indicated by a comma.
3. French Format (-- --- ---~,-- / -- --- ---~,---) : This format has spaces as the grouping character and a comma indicating the decimal place.

The following lines of code have been added for data declaration at the beginning of the Java file:

```
public static String g_dataFormat = null;
public TableSorter(){
}
```

The type *g_dataFormat* holds the user format passed from the front-end.

Removing Special Characters from the Numeric Strings

Changes are made in the *compare()* method which accepts two objects and sorts them depending on whether they are numeric, strings or dates. In this case, the numeric data is stored as strings. These strings must therefore be identified and treated as numbers while sorting. In addition to this, the user format must also be taken into account.

The strings must first be void of all the special characters (%, +, *, $), including leading and trailing spaces.

Checking whether the Strings are Numeric

The method *checkForNumber(String strObj)* checks whether string is numeric or not. If both the strings are classified as numeric, then numeric sort is performed, else they are treated as strings and sorted accordingly.

Identifying the Format of the Strings and Data Type

The method *String[] convertToFormat(String strInpObj, String strInputFormat)* converts the string to the format selected by the user.

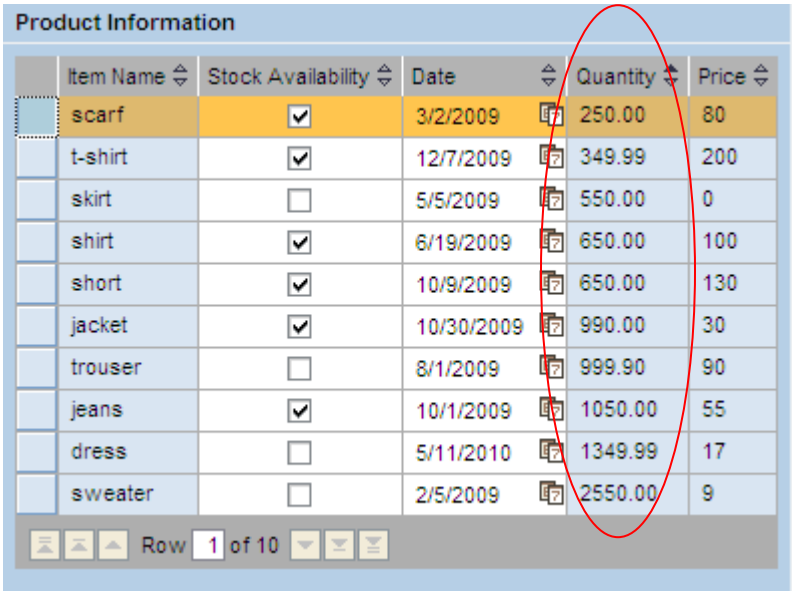
1. The method *String checkFormat(String strInputFormat)* accepts the user format from the global variable *g_dataFormat* and classifies the formats into 3 classes : us, german and french.
2. Once the format has been identified, the strings are converted into this format and categorized into 4 data types, namely *int*, *long*, *float* and *double* using the *String checkType(String strInput)* method depending on the length and presence or absence of the decimal point

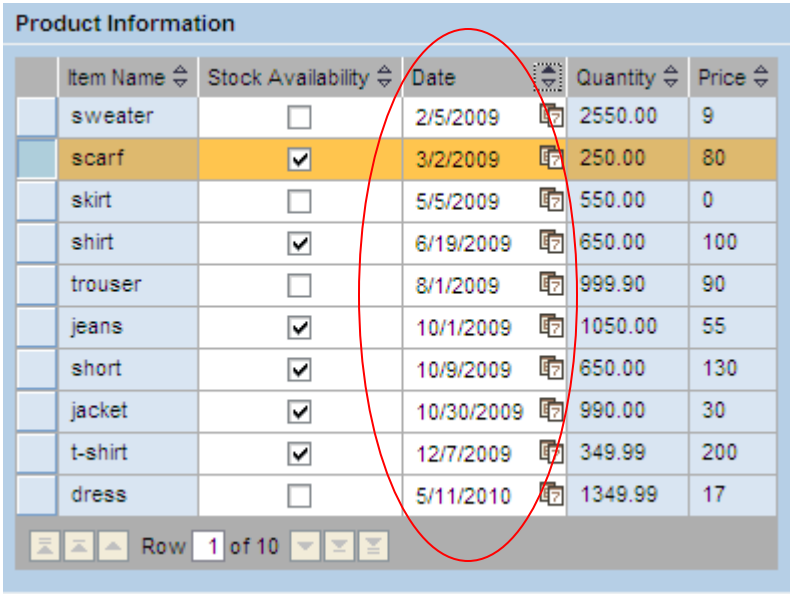
Sorting the Numeric Strings

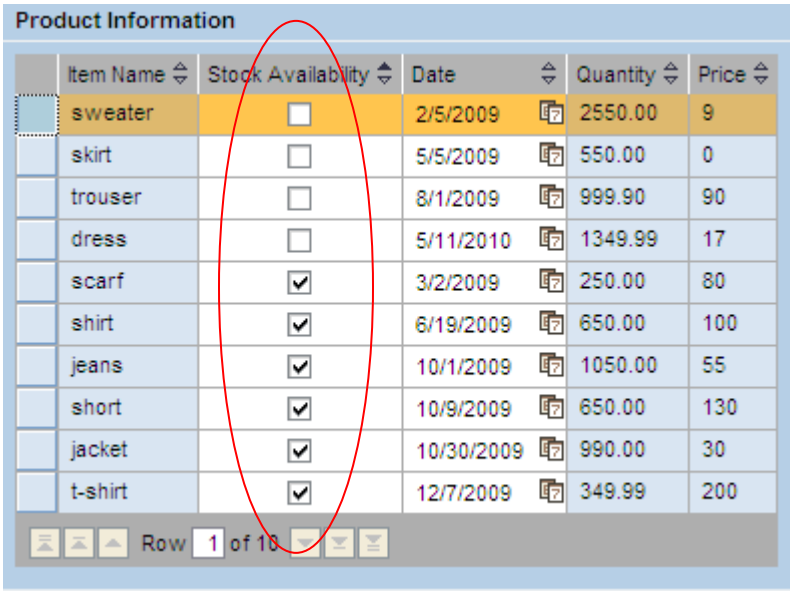
After identifying the strings as numbers, converting them into the user format and categorizing them into the respective data types, the numbers are now cast to the higher of the two data types, compared using the built-in method *compareTo()* and the result is passed to the front-end.

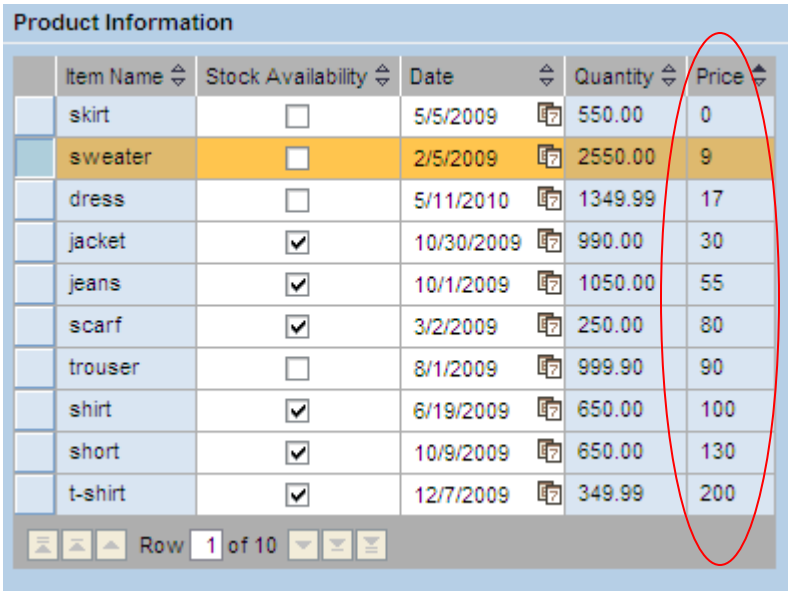
Build and Deploy and Run project

Now try sorting on the *Quantity, Stocks Availability, Date* and *Price* columns. Although these columns are of the string data type, after enhancing the sorting logic for numeric strings based on the region-specific format, the data is sorted in the correct order.




Product Information
Item Name
Stock Availability
Date
Quantity
Price
sweater 2/5/2009 2550.00 9
scarf 3/2/2009 250.00 80
skirt 5/5/2009 550.00 0
shirt 6/19/2009 650.00 100
trouser 8/1/2009 999.90 90
jeans 10/1/2009 1050.00 55
short 10/9/2009 650.00 130
jacket 10/30/2009 990.00 30
t-shirt 12/7/2009 349.99 200
dress 5/11/2010 1349.99 17
Row 1 of 10


Product Information
Item Name
Stock Availability
Date
Quantity
Price
sweater 2/5/2009 2550.00 9
skirt 5/5/2009 550.00 0
trouser 8/1/2009 999.90 90
dress 5/11/2010 1349.99 17
scarf 3/2/2009 250.00 80
shirt 6/19/2009 650.00 100
jeans 10/1/2009 1050.00 55
short 10/9/2009 650.00 130
jacket 10/30/2009 990.00 30
t-shirt 12/7/2009 349.99 200
Row 1 of 10


Product Information
Item Name
Stock Availability
Date
Quantity
Price
skirt 5/5/2009 550.00 0
sweater 2/5/2009 2550.00 9
dress 5/11/2010 1349.99 17
jacket 10/30/2009 990.00 30
jeans 10/1/2009 1050.00 55
scarf 3/2/2009 250.00 80
trouser 8/1/2009 999.90 90
shirt 6/19/2009 650.00 100
short 10/9/2009 650.00 130
t-shirt 12/7/2009 349.99 200
Row 1 of 10

TableSorter.java file

```
package com.sap.mytablesorter.tablesorter;


import java.text.Collator;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;

import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractDropDownByIndex;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractDropDownByKey;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractInputField;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractTableColumn;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDCaption;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDCheckBox;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDLink;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDProgressIndicator;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDRadioButton;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTable;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTableCellEditor;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTableColumn;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTableColumnGroup;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTextEdit;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDTextView;
import com.sap.tc.webdynpro.clientserver.uielib.standard.api.WDTableColumnSortDirection;
import com.sap.tc.webdynpro.progmodel.api.IWDAction;
import com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent;
import com.sap.tc.webdynpro.progmodel.api.IWDNode;
import com.sap.tc.webdynpro.progmodel.api.IWDNodeElement;
import com.sap.tc.webdynpro.progmodel.api.IWDViewElement;
import com.sap.tc.webdynpro.services.sal.localization.api.WDResourceHandler;

/**
 * @author Divyata Dal
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class TableSorter {
	/**
	 * @param table
	 * @param sortAction
	 * @param comparators
	 */

//		<Divyata Dal> starts

	public static String g_dataFormat = null;

	public TableSorter(){
	}

//		<Divyata Dal> ends

	public TableSorter(IWDTable table, IWDAction sortAction, Map comparators) {
		init(table, sortAction, comparators, null);
	}
```

```
	public TableSorter(IWDTable table, IWDAction sortAction, Map comparators, String[]
sortableColumns) {
		init(table, sortAction, comparators, sortableColumns);
	}

	/**
	 * Initialisation stuff
	 */
	private void init(IWDTable table, IWDAction sortAction, Map comparators, String[]
sortableColumns){
		this.table = table;
		if(sortableColumns == null){
			sortableCols = null;
		}else{
			sortableCols = new HashMap();
			for (int i = 0; i < sortableColumns.length; i++) {
				sortableCols.put(sortableColumns[i], sortableColumns[i]);
			}
		}

		// sanity checks
		if (sortAction == null)
			throw new IllegalArgumentException("Sort action must be given");
		if (table == null)
			throw new IllegalArgumentException("Table must be given");
		if (table.bindingOfDataSource() == null)
			throw new IllegalArgumentException(
				"Data source of table with id '" + table.getId() + "' must be
bound");

		// make the columns sortable
		String dataSourcePrefix = table.bindingOfDataSource() + ".";
		setComparatorsForColumns(dataSourcePrefix, table.iterateGroupedColumns(),
comparators);

		//set up the table properties
		table.setOnSort(sortAction);
		table.mappingOfOnSort().addSourceMapping(IWDTable.IWDOnSort.COL,
"selectedColumn");
		table.mappingOfOnSort().addSourceMapping(IWDTable.IWDOnSort.DIRECTION,
"sortDirection");
	}

	/**
	 * Try to make the given columns sortable (recusivly, if necessary)
	 */
	private void setComparatorsForColumns(String dataSourcePrefix, Iterator
columnIterator, Map comparators){
		int index = 0;
		for (Iterator it = columnIterator; it.hasNext(); ++index) { // for every
column: try to make it bindable
			IWDAbstractTableColumn abstractColumn = (IWDAbstractTableColumn)
it.next();
			if(abstractColumn instanceof IWDTableColumn){

				IWDTableColumn column = (IWDTableColumn)abstractColumn;
				if(sortableCols == null ||
sortableCols.containsKey(column.getId())){
					//try to make this column sortable
					Comparator comparator = null;
					if (comparators != null){
						comparator =
(Comparator)comparators.get(column.getId());
					}
```

```
                                    NodeElementByAttributeComparator elementComparator =
null;
                                    if (comparator instanceof
NodeElementByAttributeComparator) {
                                          // the easy one, attribute and ordering are given
                                          elementComparator =
(NodeElementByAttributeComparator)comparator;
                                    } else { // attribute must be determined
                                          String bindingOfPrimaryProperty =
bindingOfPrimaryProperty(column.getTableCellEditor());
                                          if (bindingOfPrimaryProperty == null ||
!bindingOfPrimaryProperty.startsWith(dataSourcePrefix)){
                                                //no attribute found or outside of data
source

      column.setSortState(WDTableColumnSortDirection.NOT_SORTABLE);
                                                continue;
                                          }
                                          String attributeName =
bindingOfPrimaryProperty.substring(dataSourcePrefix.length());
                                          Collection subnodes = new ArrayList();
                                          if (attributeName.indexOf('.') >= 0){
                                                //attribute not immediately below data
source
                                                String[] tokens = tokenize (attributeName,
".");
                                                for(int i=0; i<tokens.length-1; i++){
                                                      subnodes.add(tokens[i]);
                                                }
                                                attributeName = tokens[tokens.length-1];
                                          }
                                          if(subnodes.size() == 0){
                                                elementComparator = new
NodeElementByAttributeComparator(attributeName, comparator);
                                          }else{
                                                elementComparator = new
NodeElementByAttributeComparator(attributeName, comparator, subnodes);
                                          }
                                    }

                                    // set up internal data structures
                                    comparatorForColumn.put(column, elementComparator);

                                    //set sort state
                                    column.setSortState(WDTableColumnSortDirection.NONE);
                              }else{
                                    //column should not be sortable

      column.setSortState(WDTableColumnSortDirection.NOT_SORTABLE);
                              }

                        }else if (abstractColumn instanceof IWDTableColumnGroup){
                              //it's just a column group -> try to bind the columns of the
column group
                              IWDTableColumnGroup columnGroup =
(IWDTableColumnGroup)abstractColumn;
                              setComparatorsForColumns(dataSourcePrefix,
columnGroup.iterateColumns(), comparators);
                        }

                  }
      }

      /**
```

```
        * Tokenizes the input string according to the given delimiters. The delimiters
will be left out.
        * Example: tokenize("Hello_World", "_") results ["Hello", "World"]
        */
       private String[] tokenize (String input, String delim){
              StringTokenizer tokenizer = new StringTokenizer(input, delim);
              String[] tokens = new String[tokenizer.countTokens()];
              int index = 0;
              while(tokenizer.hasMoreTokens()){
                     tokens[index] = tokenizer.nextToken();
                     index++;
              }
              return tokens;
       }

       /**
        * This method must be called from the event handler of this table sorter's
        * sort action. It performs the actual sort operation.
        */
       public void sort(IWDCustomEvent wdEvent, IWDNode dataSource, String dataFormat) {

              g_dataFormat = dataFormat;

              // find the things we need
              String columnId = wdEvent.getString("selectedColumn");
              String direction = wdEvent.getString("sortDirection");
              IWDTableColumn column = (IWDTableColumn)
table.getView().getElement(columnId);
              NodeElementByAttributeComparator elementComparator =
(NodeElementByAttributeComparator) comparatorForColumn.get(column);

              if (elementComparator == null){
                     //not a sortable column
                     column.setSortState(WDTableColumnSortDirection.NOT_SORTABLE);
                     return;
              }

              try{
                     // sorting

       elementComparator.setSortDirection(WDTableColumnSortDirection.valueOf(direction));
                     dataSource.sortElements(elementComparator);
              }
              catch (Exception e) {
                     // TODO: handle exception.

              }

       }

       /**
        * Returns the binding of the given table cell editor's property that is
        * considered "primary" or <code>null</code> if no such binding exists or no
        * such property can be determined.
        */
       private static final String bindingOfPrimaryProperty(IWDTableCellEditor editor) {
              return editor instanceof IWDViewElement ?
bindingOfPrimaryProperty((IWDViewElement) editor) : null;
       }

       /**
        * Returns the binding of the given view element's property that is
        * considered "primary" or <code>null</code> if no such binding exists or no
        * such property can be determined.
        */
```

```
	private static final String bindingOfPrimaryProperty(IWDViewElement element) {
		if (element instanceof IWDAbstractDropDownByIndex)
			return ((IWDAbstractDropDownByIndex) element).bindingOfTexts();
		if (element instanceof IWDAbstractDropDownByKey)
			return ((IWDAbstractDropDownByKey) element).bindingOfSelectedKey();
		if (element instanceof IWDAbstractInputField)
			return ((IWDAbstractInputField) element).bindingOfValue();
		if (element instanceof IWDCaption)
			return ((IWDCaption) element).bindingOfText();
		if (element instanceof IWDCheckBox)
			return ((IWDCheckBox) element).bindingOfChecked();
		if (element instanceof IWDLink)
			return ((IWDLink) element).bindingOfText();
		if (element instanceof IWDProgressIndicator)
			return ((IWDProgressIndicator) element).bindingOfPercentValue();
		if (element instanceof IWDRadioButton)
			return ((IWDRadioButton) element).bindingOfSelectedKey();
		if (element instanceof IWDTextEdit)
			return ((IWDTextEdit) element).bindingOfValue();
		if (element instanceof IWDTextView)
			return ((IWDTextView) element).bindingOfText();

		return null;
	}

	/**
	 * Instance of a comparator according to the ordering imposed by the
	 * implementation of <code>Comparable</code>.
	 */
	private static final Comparator DEFAULT = new Comparator() {
		/**
		 * Compares the given objects according to the ordering imposed by the
first
		 * ones <code>compareTo(Object)</code> function. Furthermore,
<code>null</code>
		 * is treated to be less than any object.
		 *
		 * @see java.lang.Comparable#compareTo(java.lang.Object)
		 * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
		 */
		public int compare(Object o1, Object o2) {
			if (o1 == null && o2 == null)
				return 0;
			if (o1 == null)
				return -1;
			if (o2 == null)
				return +1;
			if (o1 instanceof Boolean && o2 instanceof Boolean)
				return o1.toString().compareTo(o2.toString()); // false < true
			if (o1 instanceof String && o2 instanceof String){

//                              <Divyata Dal> starts
			TableSorter l_tblSort = new TableSorter();
			String strObj1 = o1.toString().trim();
			String strObj2 = o2.toString().trim();
			String strTemp = "";

			String strInpForm = g_dataFormat;

			boolean flagObj1 = true;
			boolean flagObj2 = true;

			// Removing the '%' sign from numeric strings
			strObj1 = strObj1.replaceAll("%", "");
			strObj2 = strObj2.replaceAll("%", "");
```

```
//                  Removing the '+' and '*' signs from the mumeric strings
                    strTemp = "";
                    for(int i=0 ; i<strObj1.length() ; i++)
                    {
                          if(strObj1.charAt(i) != '+' && strObj1.charAt(i) != '*')
                                strTemp = strTemp + strObj1.charAt(i);
                    }
                    strObj1 = strTemp;
                    strTemp = "";
                    for(int i=0 ; i<strObj2.length() ; i++)
                    {
                          if(strObj2.charAt(i) != '+' && strObj2.charAt(i) != '*')
                                strTemp = strTemp + strObj2.charAt(i);
                    }
                    strObj2 = strTemp;

                    // Handling blank strings
                    if(strObj1.trim().length() == 0)
                          strObj1 = "0";
                    if(strObj2.trim().length() == 0)
                          strObj2 = "0";

                    // Checking whether the strings are numeric or not
                    flagObj1 = l_tblSort.checkForNumber(strObj1);
                    flagObj2 = l_tblSort.checkForNumber(strObj2);

                    // Sorting the strings if both the strings are numeric
                    if(flagObj1 && flagObj2)
                    {
                          String strNum1[] = l_tblSort.convertToFormat(strObj1,
strInpForm);
                          String strNum2[] = l_tblSort.convertToFormat(strObj2,
strInpForm);

                          if("double".compareTo(strNum1[0]) == 0 ||
"double".compareTo(strNum2[0]) == 0)
                          {
                                Double l_double1 = new Double(strNum1[1]);
                                Double l_double2 = new Double(strNum2[1]);
                                return l_double1.compareTo(l_double2);
                          }
                          else if("float".compareTo(strNum1[0]) == 0 ||
"float".compareTo(strNum2[0]) == 0)
                          {
                                Float l_float1 = new Float(strNum1[1]);
                                Float l_float2 = new Float(strNum2[1]);
                                return l_float1.compareTo(l_float2);
                          }
                          else if("long".compareTo(strNum1[0]) == 0 ||
"long".compareTo(strNum2[0]) == 0)
                          {
                                Long l_long1 = new Long(strNum1[1]);
                                Long l_long2 = new Long(strNum2[1]);
                                return l_long1.compareTo(l_long2);
                          }
                          else if("int".compareTo(strNum1[0]) == 0 &&
"int".compareTo(strNum2[0]) == 0)
                          {
                                Integer l_int1 = new Integer(strNum1[1]);
                                Integer l_int2 = new Integer(strNum2[1]);
                                return l_int1.compareTo(l_int2);
                          }
                    }
```

```
//                                <Divyata Dal> ends

                            //Use a Collator for sorting according to the given Locale
                            Collator collate =
Collator.getInstance(WDResourceHandler.getCurrentSessionLocale());
                            return collate.compare(o1, o2);
                        }
                        return ((Comparable) o1).compareTo((Comparable) o2);
                }
        };

        /**
         * Map of table column to comparator (<code>ReversableComparator</code>)
         * used for sorting that column (sortable columns only).
         */
        private Map comparatorForColumn = new HashMap();

        /**
         * The table to be sorted.
         */
        private IWDTable table = null;

        /**
         * Column-IDs of the columns, which should be sortable
         */
        private Map sortableCols = null;

//                  <Divyata Dal> starts

        /*
                This method returns a 'boolean' value indicating whether the entered string
is numeric or not
        */
        private static boolean checkForNumber(String strObj)
        {
                String strTemp = "";
                int check = 0;
                boolean flag = false;

//              Check for negative values
                if(strObj.charAt(0) == '-')
                        strObj = strObj.substring(1,strObj.length());

//              Remove blank spaces
                strObj = strObj.replaceAll(" ", "");

                if(strObj.indexOf('.') != -1 || strObj.indexOf(',') != -1)
                {
                        for(int i=0 ; i<strObj.length() ; i++)
                        {
                                if(strObj.charAt(i) != '.')
                                        strTemp = strTemp + strObj.charAt(i);
                        }
                        strTemp = strTemp.replaceAll(",", "");
                }
                else
                        strTemp = strObj.trim();

                for(int i=0 ; i<strTemp.length() ; i++)
                {
                        check = (int) strTemp.charAt(i);
                        if(check >= 48 && check <=57)
                                flag = true;
                        else
                        {
```

```
                        flag = false;
                        break;
                    }
                }

                return flag;
        }

        /*
                This method identifies the format passed comparing it with standard format
        */
        private static String checkFormat(String strInputFormat)
        {
                String strFormat = "";

                if("--,---,---~.--".compareTo(strInputFormat) == 0 || "--,---,---~.---
".compareTo(strInputFormat) == 0)
                        strFormat = "us";
                else if("--.---.---~,--".compareTo(strInputFormat) == 0 || "--.---.---~,---
".compareTo(strInputFormat) == 0)
                        strFormat = "german";
                else if("-- --- ---~,--".compareTo(strInputFormat) == 0 || "-- --- ---~,---
".compareTo(strInputFormat) == 0)
                        strFormat = "french";

                return strFormat;
        }

        /*
                 This method classifies the string as int, long, float or double once
converted into the system format
        */
        private static String checkType(String strInput)
        {
                String strType = "";

                if(strInput.length() < 9)
                {
                        if(strInput.indexOf('.') == -1)
                                strType = "int";
                        else
                                strType = "float";
                }
                else if(strInput.length() >= 9)
                {
                        if(strInput.indexOf('.') == -1)
                                strType = "long";
                        else
                                strType = "double";
                }

                return strType;
        }

        /*
                This method converts the string in the system format
        */
        private static String[] convertToFormat(String strInpObj, String strInputFormat)
        {
                String strFormat = "";
                String[] strDet = new String[2];
                String strTemp = "";
                int len = 0;

                strFormat = checkFormat(strInputFormat);
```

```
            if("us".compareTo(strFormat) == 0)
            {
                  strDet[1] = strInpObj.trim().replaceAll(",", "");
                  strDet[0] = checkType(strDet[1]);
            }

            else if("german".compareTo(strFormat) == 0)
            {
                  for(int i=0 ; i<strInpObj.length() ; i++)
                  {
                        if(strInpObj.charAt(i) != '.')
                              strTemp = strTemp + strInpObj.charAt(i);
                  }
                  strDet[1] = strTemp.trim().replaceAll(",", ".");
                  strDet[0] = checkType(strDet[1]);
            }

            else if("french".compareTo(strFormat) == 0)
            {
                  strInpObj = strInpObj.replaceAll(" ", "");
                  strDet[1] = strInpObj.trim().replaceAll(",", ".");
                  strDet[0] = checkType(strDet[1]);
            }

                  return strDet;
            }

//                  <Divyata Dal> ends

      /**
       * Generic comparator that compares node elements by a given attribute with
       * the help of a given comparator.
       */
      public final class NodeElementByAttributeComparator implements Comparator {


            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute according to the natural ordering of that attribute's
             * type (which must implement <code>java.lang.Comparable</code>).
             */
            public NodeElementByAttributeComparator(String attributeName) {
                  this(attributeName, null, false, new ArrayList());
            }

            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute with the help of the given comparator. If no
comparator
             * is given, the natural ordering of that attribute's type is used.
             */
            public NodeElementByAttributeComparator(String attributeName, Comparator
comparator) {
                  this(attributeName, comparator, false, new ArrayList());
            }

            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute either as objects (i.e. "in internal format") or as
text
             * (i.e. "in external format") as indicated. The ordering is the natural
```

```
             * ordering of that attribute's type (which must implement
             * <code>java.lang.Comparable</code>) in case objects are compared or the
             * natural ordering of <code>java.lang.String</code> in case texts are
compared.
             */
            public NodeElementByAttributeComparator(String attributeName, boolean
compareAsText) {
                  this(attributeName, null, compareAsText, new ArrayList());
            }

            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute according to the natural ordering of that attribute's
             * type (which must implement <code>java.lang.Comparable</code>). In
addition it is possible
             * to define the path to a child node with the
<code>java.util.Collection</code> subnodes.
             * (List of child node names in the correct order)
             */
            public NodeElementByAttributeComparator(String attributeName, Collection
subnodes) {
                  this(attributeName, null, false, subnodes);
            }

            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute with the help of the given comparator. If no
comparator
             * is given, the natural ordering of that attribute's type is used. In
addition it is possible
             * to define the path to a child node with the
<code>java.util.Collection</code> subnodes.
             * (List of child node names in the correct order)
             */
            public NodeElementByAttributeComparator(String attributeName, Comparator
comparator, Collection subnodes) {
                  this(attributeName, comparator, false, subnodes);
            }

            /**
             * Creates a new comparator for the given attribute name that compares
values
             * of that attribute either as objects (i.e. "in internal format") or as
text
             * (i.e. "in external format") as indicated. The ordering is the natural
             * ordering of that attribute's type (which must implement
             * <code>java.lang.Comparable</code>) in case objects are compared or the
             * natural ordering of <code>java.lang.String</code> in case texts are
compared. In addition it is possible
             * to define the path to a child node with the
<code>java.util.Collection</code> subnodes.
             * (List of child node names in the correct order)
             */
            public NodeElementByAttributeComparator(String attributeName, boolean
compareAsText, Collection subnodes) {
                  this(attributeName, null, compareAsText, subnodes);
            }

            /**
             * Internal constructor.
             */
            private NodeElementByAttributeComparator(
                  String attributeName,
```

```
                    Comparator comparator,
                    boolean compareAsText,
                    Collection subNodes) {
                    if (attributeName == null)
                           throw new IllegalArgumentException("Attribute name must not be
null");
                    if (comparator == null)
                           comparator = DEFAULT;

                    this.attributeName = attributeName;
                    this.comparator = comparator;
                    this.compareAsText = compareAsText;
                    this.sortDirection = true;
                    this.subNodes = subNodes;
             }

             /**
              * Sets the sort direction of this comparator to the given direction. The
comparator sort in ascending order by default.
              * @see
com.sap.tc.webdynpro.clientserver.uielib.standard.api.WDTableColumnSortDirection
              */
             public void setSortDirection(WDTableColumnSortDirection direction){
                    if(direction.equals(WDTableColumnSortDirection.UP)){
                           sortDirection = true;
                    }else if(direction.equals(WDTableColumnSortDirection.DOWN)){
                           sortDirection = false;
                    }
             }

             /**
              * Compares the given objects which must be instances of
<code>IWDNodeElement</code>
              * according to the values of the attribute given at construction time
              * with the help of the comparator given at construction time.
              *
              * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
              * @see com.sap.tc.webdynpro.progmodel.api.IWDNodeElement
              */
             public int compare(Object o1, Object o2) {
                    IWDNodeElement element1 = (IWDNodeElement) o1;
                    IWDNodeElement element2 = (IWDNodeElement) o2;
                    if(subNodes.size() > 0){
                           element1 = getSubNodeElement(element1, 0);
                           element2 = getSubNodeElement(element2, 0);
                    }
                    Object attributeValue1 = null;
                    Object attributeValue2 = null;
                    if(element1 != null){
                           attributeValue1 =
                                  compareAsText
                                         ? element1.getAttributeAsText(attributeName)
                                         : element1.getAttributeValue(attributeName);
                    }
                    if(element2 != null){
                           attributeValue2 =
                                  compareAsText
                                         ? element2.getAttributeAsText(attributeName)
                                         : element2.getAttributeValue(attributeName);
                    }

                    if(sortDirection){
                           return comparator.compare(attributeValue1, attributeValue2);
                    }else{
                           return comparator.compare(attributeValue2, attributeValue1);
```

```
                  }
            }

            /**
             * Determines recursivly the child node, which have an attribute with the
given name.
             * The path to this child node must be specified in the subnodes property
of this comparator.
             * Start this method with index = 0.
             */
            private IWDNodeElement getSubNodeElement(IWDNodeElement currentElement, int
index){
                  if(currentElement == null || index >= subNodes.size()){
                        //end of recursion
                        return currentElement;
                  }else{
                        return
getSubNodeElement(currentElement.node().getChildNode((String)subNodes.toArray()[index],
currentElement.index()).getCurrentElement(), index+1);
                        //return
getSubNodeElement(currentElement.node().getChildNode((String)subNodes.toArray()[index],
currentElement.index()).getElementAt(0), index+1);
                  }
            }

            /**
             * Name of the attribute used for comparisons.
             */
            private final String attributeName;

            /**
             * Comparator used for comparing the attribute's values.
             */
            private final Comparator comparator;

            /**
             * Indicates whether attribute values are compared as text (as opposed to
             * "as objects").
             */
            private final boolean compareAsText;

            /**
             * Sort direction (true = ascending order, false = descending order)
             */
            private boolean sortDirection;

            /**
             * List of child node names
             * (Description of the path from the given context node to the specified
attribute)
             */
            private Collection subNodes;
      }

}
```

Related Content

Table Sorter Java file

Implementing a Table with Sorting and Filtering Capabilities

Developing with Tables in Web Dynpro

For more information, visit the User Interface Technology homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.