GRIDCOLHEADER - Flexible Column Headers

In the example introducing the ROWTABLEAREA2 control, the header of the grid was built by arranging certain LABEL controls, where the LABEL controls where rendered as headers:

```
<rowtablearea2 griddataprop="lines" rowcount="10" withborder="true" width="100%">
    <tr>
        ...
        <label name="First Name" asheadline="true">
        </label>
        ...
    </tr>
    <repeat>
...
...
...
```

It is also possible to use the GRIDCOLHEADER control in order to define the header of a grid. The advantages are:

- GRIDCOLHEADER controls are automatically rendered in "header style".
- GRIDCOLHEADER controls allow to sort the grid content.
- GRIDCOLHEADER controls allow to resize a grid.
- GRIDCOLHEADER controls allow to change the sequence of columns - if used together with the FLEXLINE control.

This chapter covers the following topics:

- Flexible Column Sizing
- Flexible Column Sorting
- Flexible Column Sequence
- GRIDCOLHEADER Properties
- Smart Selection of Rows - SELECTOR Control
- SELECTOR Properties

Flexible Column Sizing

Let us have a look on the following grid definition:

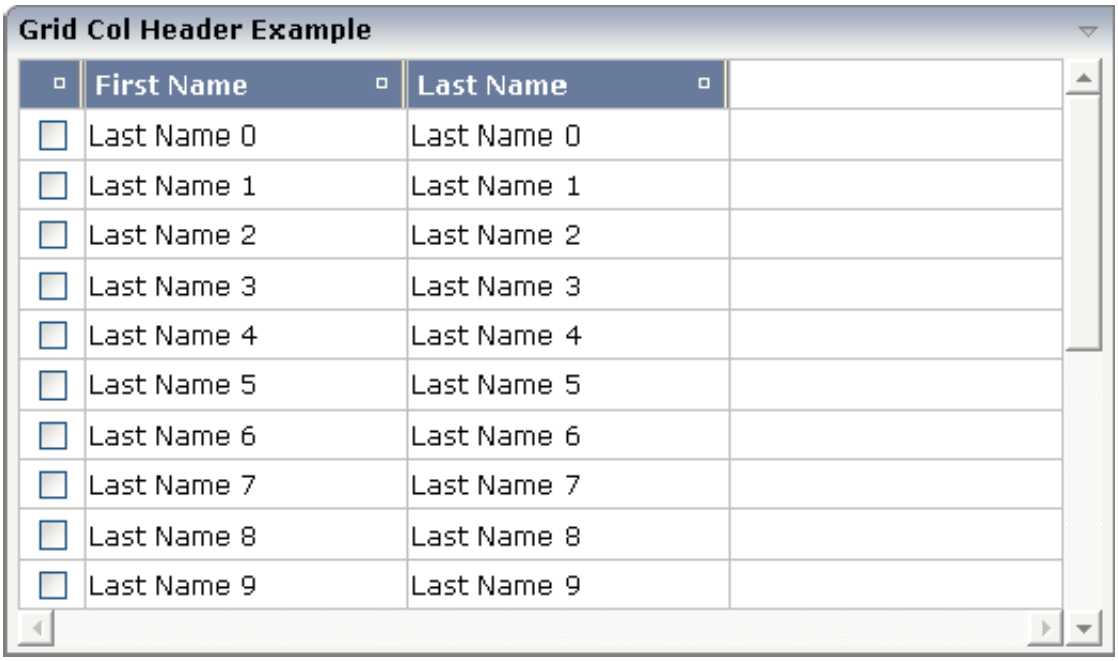
```
<rowarea name="Grid Col Header Example">
    <rowtablearea2 griddataprop="lines" rowcount="10" width="100%" withborder="true"
                   hscroll="true" firstrowcolwidths="true">
        <tr>
            <gridcolheader name=" " width="30">
```

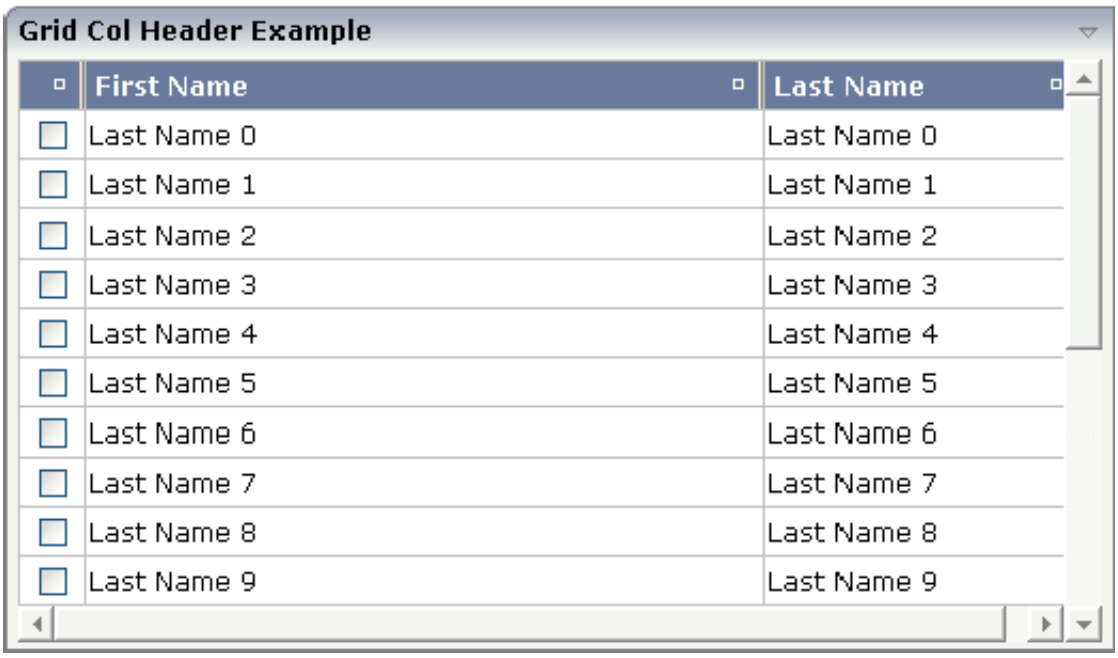
```
            </gridcolheader>
            <gridcolheader name="First Name" width="150">
            </gridcolheader>
            <gridcolheader name="Last Name" width="150">
            </gridcolheader>
            <hdist>
            </hdist>
        </tr>
        <repeat>
            <str valueprop="selected">
                <checkbox valueprop="selected" flush="screen" width="100%" align="center">
                </checkbox>
                <field valueprop="firstName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <field valueprop="lastName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <hdist>
                </hdist>
            </str>
        </repeat>
    </rowtablearea2>
</rowarea>
```

You see:

- The ROWTABLEAREA2 definition was set to always follow the column widths of the first row. The first row of the grid is the row containing the GRIDCOLHEADER controls, this means that this row defines the column sizing for the whole grid.
- The header row of the grid is built out of GRIDCOLHEADER controls, each one specifying a name and a width.
- The header row is closed with an horizontal distance.This is quite important: if your column widths do not horizontally fill the grid, then the remaining space is typically equally distributed among the columns. Even if GRIDCOLHEADER specifies a certain width, this may still be overridden by the browser. A horizontal distance control (HDIST) at the end makes the browser assign the remaining space to the distance control, not to the GRIDCOLHEADER controls.

When the user moves the mouse over the border of the header columns, then the cursor will change and the user can change the width of the columns:





Your adapter gets notified by a certain event if the user changes the width of the columns. Have a look at the adapter code:

```
// This class is a generated one.

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.GRIDCOLHEADERInfo;
import com.softwareag.cis.server.util.GRIDCollection;
import com.softwareag.cis.server.util.IGRIDCOLHEADERChangeListener;
import com.softwareag.cis.server.util.ISSSARRAYInfo;

public class FlexibleColumnSizingAdapter
    extends Adapter
    implements IGRIDCOLHEADERChangeListener
```

```
{
    // class >LinesItem<
    public class LinesItem
    {
        // property >firstName<
        String m_lastName;
        public String getLastName() { return m_firstName; }
        public void setLastName(String value) { m_firstName = value; }

        String m_firstName;
        public String getFirstName() { return m_firstName; }
        public void setFirstName(String value) { m_firstName = value; }

        // property >selected<
        boolean m_selected;
        public boolean getSelected() { return m_selected; }
        public void setSelected(boolean value) { m_selected = value; }
    }

    // property >lines<
    GRIDCollection m_lines = new GRIDCollection();
    public GRIDCollection getLines() { return m_lines; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        for ( int i=0; i<20; i++)
        {
            LinesItem line = new LinesItem();
            line.setFirstName("First Name " +i);
            line.setLastName("Last Name " + i);
            m_lines.add(line);
        }
        m_lines.addGridColHeaderChangeListener(this);

    }

//  ------------------------------------------------------------------------
//  interface IGRIDCOLHEADERChangeListener
//  ------------------------------------------------------------------------
    public void reactOnContextMenuRequest(ISSSARRAYInfo collection,
            GRIDCOLHEADERInfo colInfo)
    {  }

    public void reactOnMove(ISSSARRAYInfo collection, GRIDCOLHEADERInfo[] colInfo)
    {  }

    public void reactOnResize(ISSSARRAYInfo collection,
            GRIDCOLHEADERInfo[] colInfos)
    {
        String colWidths = "";
        for (int i=0; i<colInfos.length; i++)
        {
            colWidths += colInfos[i].getWidth();
            colWidths += " ";
        }
        outputMessage(MT_SUCCESS,colWidths);
    }
}
```

The interface `IGRIDCOLHEADERChangeListener` passes all events inside the GRIDCOLHEADER controls to the adapter code. You register the interface by using the `GRIDCollection`'s `addColHeaderChangeListener` method.

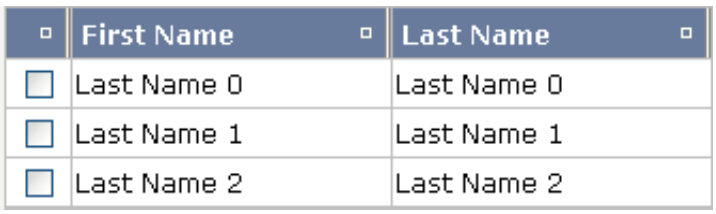
In addition, you can set the widths of the columns by your adapter, e.g. you may store column widths inside your application in order to save the user's column settings and later on reapply the width information.

Flexible Column Sorting

The GRIDCOLHEADER allows to bind to a property which is used for sorting. The XML definition of the previous example was extended to demonstrate this:

```
<rowarea name="Grid Col Header Example">
    <rowtablearea2 griddataprop="lines" rowcount="10" width="100%" withborder="true"
                   hscroll="true" firstrowcolwidths="true">
        <tr>
            <gridcolheader name=" " width="30" propref="selected">
            </gridcolheader>
            <gridcolheader name="First Name" width="150" propref="firstName">
            </gridcolheader>
            <gridcolheader name="Last Name" width="150" propref="lastName">
            </gridcolheader>
            <hdist>
            </hdist>
        </tr>
        <repeat>
            <str valueprop="selected">
                <checkbox valueprop="selected" flush="screen" width="100%" align="center">
                </checkbox>
                <field valueprop="firstName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <field valueprop="lastName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <hdist>
                </hdist>
            </str>
        </repeat>
    </rowtablearea2>
</rowarea>
```

Each GRIDCOLHEADER control now points to the property that is referenced in the subsequent FIELD/CHECKBOX definition. The control now displays small sort icons. The user can sort the information by choosing the icon. Sorting is done implicitly on the server side, i.e. the order of items inside the collection is changed without any programming effort.



The automated sorting is a feature of the GRID collection object. The sorting is data type aware, i.e. if a property is a string property, then lexical sorting is performed; if it is, for example, an integer or float property, then numeric sorting is performed.

Flexible Column Sequence

Let us have a look at the following grid definition:

```
<rowtablearea2 griddataprop="lines" rowcount="20" width="100%"
               withborder="true" hscroll="true" firstrowcolwidths="true">
   <tr>
     <label width="25" asheadline="true">
     </label>
     <flexline infoprop="gridheaderline">
     </flexline>
     <hdist>
     </hdist>
   </tr>
   <repeat>
      <str valueprop="selected">
         <selector valueprop="selected">
         </selector>
         <flexline infoprop="/flexInfo">
         </flexline>
         <hdist>
         </hdist>
      </str>
  </repeat>
</rowtablearea2>
```

You see:

- The ROWTABLEAREA2 definition is set to always follow the column widths of the first row (FIRSTROWCOLWIDTH is set to "true"). The first row of the grid is the row containing the header FLEXLINE control. This means that this row defines the column sizing for the whole grid.
- The header row of the grid is built using a FLEXLINE control. At runtime, we pass GRIDCOLHEADER controls, where each GRIDCOLHEADER control specifies a name and a width.
- The header row is closed with a horizontal distance.This is important: if your column widths do not horizontally fill the grid, then the remaining space is typically equally distributed among the columns. Even if GRIDCOLHEADER specifies a certain width, this may still be overridden by the browser. A horizontal distance control (HDIST) at the end makes the browser assign the remaining space to the distance control, not to the GRIDCOLHEADER controls.

The user can change the sequence of the columns by moving a column header to the position of another header. Example:

	AAA	BBB	CCC	DDD
☐	1000	999	998	997
☐	996	995	994	993
☐	992	991	990	989
☐	988	987	986	985
☐	984	983	982	981
☐	980	979	978	977
☐	976	975	974	973
☐	972	971	970	969
☐	968	967	966	965

	AAA	BBB	CCC	DDD
☐	1000	999	998	BBB
☐	996	995	994	
☐	992	991	990	
☐	988	987	986	
☐	984	983	982	
☐	980	979	978	
☐	976	975	974	
☐	972	971	970	
☐	968	967	966	

	AAA	CCC	BBB	DDD
☐	1000	998	999	997
☐	996	994	995	993
☐	992	990	991	989
☐	988	986	987	985
☐	984	982	983	981
☐	980	978	979	977
☐	976	974	975	973
☐	972	970	971	969
☐	968	966	967	965

Your adapter gets notified by a certain event if the user changes the width of the columns. Have a look at the adapter code:

```
// This class is a generated one.
package com.softwareag.cis.test40;

import java.util.Hashtable;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.util.FLEXLINEInfo;
import com.softwareag.cis.server.util.GRIDCOLHEADERInfo;
import com.softwareag.cis.server.util.GRIDCollection;
import com.softwareag.cis.server.util.IGRIDCOLHEADERChangeListener;
import com.softwareag.cis.server.util.ISSSARRAYInfo;
import com.softwareag.cis.server.util.MENUNODEInfo;
import com.softwareag.cis.server.util.SelectableLine;
import com.softwareag.cis.server.util.TREECollection;

/**
 * Adapter class for Demo "/cisdemos/35_gridheader.xml/"
 * */
public class GridColHeadersAdapter
    extends Adapter
    implements IGRIDCOLHEADERChangeListener
{
    // -------------------------------------------------------------------------
    // members
    // -------------------------------------------------------------------------

    private GRIDCOLHEADERInfo[] m_gridColHeaderInfos;
    private int s_cellCounter = 1000;
    private String m_columnSortTooltip;
    private String[] m_proprefs = new String[]
    {
           "aaa",
           "bbb",
           "ccc",
           "ddd"
    };
    private String[] m_widths = new String[]
    {
          "25%",
          "25%",
          "25%",
          "25%"
    };
    private String[] m_titles = new String[]
    {
         "AAA",
         "BBB",
         "CCC",
         "DDD"
    };

    // -------------------------------------------------------------------------
    // inner classes
    // -------------------------------------------------------------------------

    /** Represents one line within the grid. The column values of the grid
     * line is kept dynamically (interface IDynamicAccess) - there are
     * no explicit Getter and Setter methods. The definition of the column
     * are kept dynamically, too (FLEXLINEInfo).
     */
    public class Line extends SelectableLine implements IDynamicAccess
    {
        Hashtable m_propertyValues = new Hashtable();


        /** Constructs a grid line that visualizes a person.*/
        public Line(String[] propRefs, String[] widths)
        {
            for (int i = 0; i < propRefs.length; i++)
                m_propertyValues.put(propRefs[i], ""+s_cellCounter--);

            rebuildContentLine(propRefs, widths);
        }

        public void rebuildContentLine(String[] propRefs,
                                       String[] width)
        {
            m_flexInfo.clear();
            for (int i = 0; i < propRefs.length; i++)
                m_flexInfo.addField(GridColHeadersAdapter.this,"valueprop;"+propRefs[i]+";" +
                                    "width;"+width[i]+";" +
                                    "noborder;true;" +
                                     "transparentbackground;true;");
        }

        /** Dynamic person properties */
        public String[] findDynamicAccessProperties()
        {
            return m_proprefs;
        }

        /** Java Datatype */
        public Class getClassForProperty(String property)
        {
            return null; // ==> String
        }
        public Object getPropertyValue(String propertyName)
        {
```

```
        return m_propertyValues.get(propertyName);
    }
    public void setPropertyValue(String propertyName, Object value)
    {
        if (value == null) value = "";
        m_propertyValues.put(propertyName, value);
    }
    public void invokeMethod(String methodName)
    {
        // nothing to do
    }
}

// ----------------------------------------------------------------------
// property access
// ----------------------------------------------------------------------

// property >gridheaderline<
FLEXLINEInfo m_gridheaderline = new FLEXLINEInfo();
public FLEXLINEInfo getGridheaderline() { return m_gridheaderline; }
public void setGridheaderline(FLEXLINEInfo value) { m_gridheaderline = value; }

// property >flexInfo<
FLEXLINEInfo m_flexInfo = new FLEXLINEInfo();
public FLEXLINEInfo getFlexInfo() { return m_flexInfo; }

// property >lines<
GRIDCollection m_lines = new GRIDCollection();
public GRIDCollection getLines() { return m_lines; }

// ----------------------------------------------------------------------
// public methods
// ----------------------------------------------------------------------

/** Is called on page load*/
public void init()
{
    m_lines.registerGridColHeaderChangeListener(this);
    m_gridColHeaderInfos = new GRIDCOLHEADERInfo[m_proprefs.length];
    for (int i = 0; i < m_proprefs.length; i++)
        m_gridColHeaderInfos[i] =  new GRIDCOLHEADERInfo(i,m_proprefs[i],m_widths[i]);
    for (int i = 0; i < 50; i++)
        m_lines.add(new Line(m_proprefs,m_widths ));
    rearrangeGridColumns(m_gridColHeaderInfos);
}

/** Is called when user re-orders columns by drag and drop */
public void reactOnMove(ISSSARRAYInfo collection, GRIDCOLHEADERInfo[] colInfo)
{
    rearrangeGridColumns(colInfo);

    // output info
    String info = replaceLiteral("release40","gridcolhead.columnorder");
    for (int i = 0; i < colInfo.length; i++)
    {
        info += replaceLiteral("release40","gridcolhead.column")+colInfo[i].getPropref().toUpperCase();
        if (i != (colInfo.length-1))
            info += ", ";
    }
    outputMessage(MT_SUCCESS, info);
}

/** Is called when user re-orders columns by drag and drop */
public void reactOnResize(ISSSARRAYInfo collection, GRIDCOLHEADERInfo[] colInfo)
{
    rearrangeGridColumns(colInfo);

    // output info
    String info = replaceLiteral("release40","gridcolhead.columnwidth");
    for (int i = 0; i < colInfo.length; i++)
    {
        info += replaceLiteral("release40","gridcolhead.column")+colInfo[i].getPropref().toUpperCase()+" ("+colInfo[i].getWidth()+")";
        if (i != (colInfo.length-1))
            info += ", ";
    }
    outputMessage(MT_SUCCESS, info);
}

/** */
public void reactOnContextMenuRequest(ISSSARRAYInfo collection, GRIDCOLHEADERInfo colInfo)
{
}

// ----------------------------------------------------------------------
// private helpers
// ----------------------------------------------------------------------

private void rearrangeGridColumns(GRIDCOLHEADERInfo[] colInfo)
{
    m_gridColHeaderInfos = colInfo;

    // HEADER
    m_gridheaderline.clear();
    for (int i = 0; i < colInfo.length; i++)
        m_gridheaderline.addGridColHeader(this,
                                          m_lines,
                                          "name;"+findTitleForPropRef(colInfo[i].getPropref())+";" +
                                          "width;"+colInfo[i].getWidth()+";" +
                                          "propref;"+colInfo[i].getPropref()+";"+
```

```
                                        "sorttitle;"+m_columnSortTooltip);

        // CONTENT
        String[] columns = new String[colInfo.length];
        for (int i = 0; i < columns.length; i++)
            columns[i] = colInfo[i].getPropref();


        String[] witdhs = new String[colInfo.length];
        for (int i = 0; i < columns.length; i++)
            witdhs[i] = colInfo[i].getWidth();

        for (int i=0; i<m_lines.size(); i++)
        {
            Line l = (Line)m_lines.get(i);
            l.rebuildContentLine(columns, witdhs);
        }
    }

    private String findTitleForPropRef(String propRef)
    {
        for (int i = 0; i < m_proprefs.length; i++)
        {
            if (m_proprefs[i].equals(propRef))
                return m_titles[i];
        }
        return ""; // should not happen
    }
}
```

The interface `IGRIDCOLHEADERChangeListener` passes all events inside the GRIDCOLHEADER controls to the adapter code. You register the interface by using the `GRIDCollection`'s `addColHeaderChangeListener` method.

In addition, you can set the sequence of the columns by your adapter, e.g. you may store the column sequence inside your application in order to save the user's column settings and later on reapply the width information.

GRIDCOLHEADER Properties

Basic			
name	Text that is displayed inside the control. Please do not specify the name when using the multi language management - but specify a "textid" instead.	Sometimes obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Sometimes obligatory	

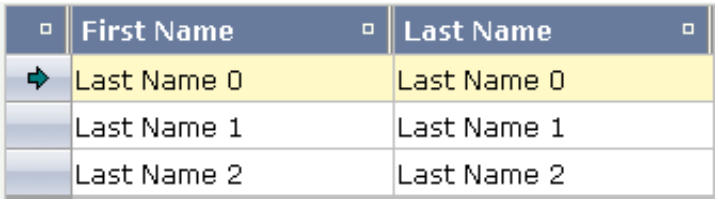
width	Width of the control. There are three possibilities to define the width: (A) You do not define a width at all. In this case the width of the control will either be a default width or - in case of container controls - it will follow the width that is occupied by its content. (B) Pixel sizing: just input a number value (e.g. "100"). (C) Percentage sizing: input a percantage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a width this control can reference. If you specify this control to have a width of 50% then the parent element (e.g. an ITR-row) may itself define a width of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Obligatory	100 120 140 160 180 200 50% 100%
propref	If the grid column visualizes data input the name of the property here. This property is located within the row item class. Example: if you use a FIELD or CHECKBOX control input the value of property VALUEPROP here. If the grid column does not visualize any data (e.g. you use a BUTTON control) input an unique column identifier. The PROPREF property is used as key when flushing ’column change events’ to the application.	Optional	
Appearance			
title	Text that is shown as tooltip for the control. Either specify the text "hard" by using this TITLE property - or use the TITLETEXTID in order to define a language dependent literal.	Optional	
titletextid	Text ID that is passed to the multi lanaguage management - representing the tooltip text that is used for the control.	Optional	
withsorticon	Flag that indicates if a small sort indicator is shown within the right corner of the control. Default is TRUE.	Optional	true false
image	URL of image that is displayed inside the control. Any image type (.gif, .jpg, ...) that your browser does understand is valid. Use the following options to specify the URL: (A) Define the URL relative to your page. Your page is generated directly into your project’s folder. Specifiying "images/xyz.gif" will point into a directory parallel to your page. Specifying "../HTMLBasedGUI/images/new.gif" will point to an image of a neighbour project. (B) Define a complete URL, like "http://www.softwareag.com/images/logo.gif".	Optional	

nowrap	The textual content of the header is not wrapped automatically. No line break will be performed automatically by the browser. If you want the text of the header to be wrapped, set the value to "false".	Optional	true false
stylevariant	Some controls offer the possibility to define style variants. By this style variant you can address different styles inside your style sheet definition file (.css). If not defined "normal" styles are chosen, if defined (e.g. "VAR1") then other style definitions (xxxVAR1xxx) are chosen. Purpose: you can set up style variants in the style sheet defintion and use them multiple times by addressing them via the "stylevariant" property. CIS currently offerst two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2 VAR3 VAR4
sorttitle	Text that is shown as tooltip for the sort indicator. Either input text by using this SORTTITLE property - or use the SORTTITLETEXTID in order to define a language dependent literal.	Optional	
sorttitletextid	Text ID that is passed to the multi lanaguage management - representing the tooltip text for the sort indicator.	Optional	
textalign	Alignment of text inside the control.	Optional	left center right
tabindex	Index that defines the tab order of the control. Controls are selected in increasing index order and in source order to resolve duplicates.	Optional	-1 0 1 2 5 10 32767

rowspan	Row spanning of control. If you use TR table rows then you may sometimes want to control the number of rows your control occupies. By default it is "1" - but you may want to define the control two span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value
colspan	Column spanning of control. If you use TR table rows then you may sometimes want to control the number of columns your control occupies. By default it is "1" - but you may want to define the control to span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value
Binding			
Comment			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor’s tree view.	Optional	

Smart Selection of Rows - SELECTOR Control

By using the SELECTOR control in combination with the STR control, you can build nice looking grids in which the user can select rows - without effort on the programming side. Have a look at the following screen:



The SELECTOR control is typically is used in the leftmost column. The user can select the control with the mouse or keyboard. In case of using the control for multiple selections, the user can select mulitple rows using a combination of CTRL and click or SHIFT and click.

The SELECTOR control references a boolan property inside a row object that is representing the selection state. The XML layout definition looks as follows:

```
<rowtablearea2 griddataprop="lines" rowcount="10" width="100%" withborder="true"
               hscroll="true" firstrowcolwidths="true">
     <tr>
         <gridcolheader name=" " width="30" propref="selected">
         </gridcolheader>
         <gridcolheader name="First Name" width="150" propref="firstName">
         </gridcolheader>
         <gridcolheader name="Last Name" width="150" propref="lastName">
         </gridcolheader>
         <hdist>
         </hdist>
     </tr>
     <repeat>
         <str valueprop="selected">
             <selector valueprop="selected" width="30" withlinenum="false"
                       singleselect="false">
             </selector>
             <field valueprop="firstName" width="100%" noborder="true"
                    transparentbackground="true">
             </field>
             <field valueprop="lastName" width="100%" noborder="true"
                    transparentbackground="true">
             </field>
             <hdist>
             </hdist>
         </str>
     </repeat>
</rowtablearea2>
```

You see the following:

- STR and SELECTOR are referencing the same property `selected` so that selections done by the SELECTOR control are automatically reflected in the selections of the row.
- SELECTOR is switched to allow multiple selections.
- By using the property `withlinenum`, you specify that inside the selector no line number is output. Instead, the SELECTOR is left empty if not selected, or it displays an icon if selected.

The selector simplifies programming of the grid selection a lot. When clicking the selector control, it automatically manages the referenced selection property of all rows that are managed inside the corresponding grid collection.

SELECTOR Properties

Basic			
valueprop	Name of adapter property that is indicating the selection status of the row that the selector refers to. The property is set and get by the SELECTOR control.	Optional	

width	Width of the control. There are three possibilities to define the width: (A) You do not define a width at all. In this case the width of the control will either be a default width or - in case of container controls - it will follow the width that is occupied by its content. (B) Pixel sizing: just input a number value (e.g. "100"). (C) Percentage sizing: input a percantage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a width this control can reference. If you specify this control to have a width of 50% then the parent element (e.g. an ITR-row) may itself define a width of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 120 140 160 180 200 50% 100%
singleselect	Indicates if the multiple lines can be selected ("false") or only one line can be selected ("true"). Default is "true".	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor’s tree view.	Optional	
Binding			
valueprop	(already explained above)		
Appearance			
withlinenum	There are two usage variants: either the line number of the corresponding row is shown as content of the SELECTOR control ("true") - or nothing is shown inside ("false"). In case of selecting "true" then the line number is automatically retrieved, i.e. you do not have to specify a property on adapter side to indicate the value of the line number.	Optional	true false
image	If specifying WITHLINENUM to be "false" then a small arrow icon is shown inside the control if selecting a corresponding row. Input the URL of the icon to be shown if you do not want to use the default icon. If specifying WITHLINENUM to be "true" then the line number of selected lines is output in bold font.	Optional	
imageprop	The URL of the image to be shown for displaying selected rows is not hard wires via the IMAGE property but "soft wired": you refer an adapter property that dynamically passes the URL of the image to be shown.	Optional	

alwaysshowicon	Flag that indicates if the selector shows its image - independent from whether the corresponding line is selected or not. With ALWAYSHOWICON you can show icons on unselected lines, too. For that specify WITHLINENUM to be "false" and use IMAGEPROP. Default is "false".	Optional	true false
tabindex	Index that defines the tab order of the control. Controls are selected in increasing index order and in source order to resolve duplicates.	Optional	-1 0 1 2 5 10 32767
Miscellaneous			
testtoolid	Use this attribute to assign a fixed control identifier that can be later on used within your test tool in order to do the object identification	Optional	