Natural for Ajax

Java Page Layout

Version 9.3.3

October 2025

ADABAS & NATURAL

This document applies to Natural for Ajax Version 9.3.3 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: NJX-NATNJX-PAGELAYOUT-JAVA-933-20251001

Table of Contents

Java Page Layout

Page Layout and Control Reference	
Typical Page Layout	Describes the elements used for the layout of a page.
Working with Containers	Shows you how to work with containers - containers are areas on the page that can hold controls.
Working with Controls	Shows you how to work with the elements that are placed into containers - the controls.
Working with Grids	Explains what grids are and how to use them.
Working with Trees	Explains the basic types of trees and how to use them.
Working with Menus	Shows you how to arrange a number of functions in a structured way.
Non-Visual Controls and Hot Keys	Describes how to develop controls that do not have visual effects.
Controls for Absolute Positioning	A special set of controls available to position them by defining their absolute x- and y-coordinates
Working with Pages	Deals with more complex applications in which you have sequences of pages.
Working with PDF Documents	PDF services: Ideas from HTML-based GUIs, transferred into the generation of PDF output

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://techcommunity.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

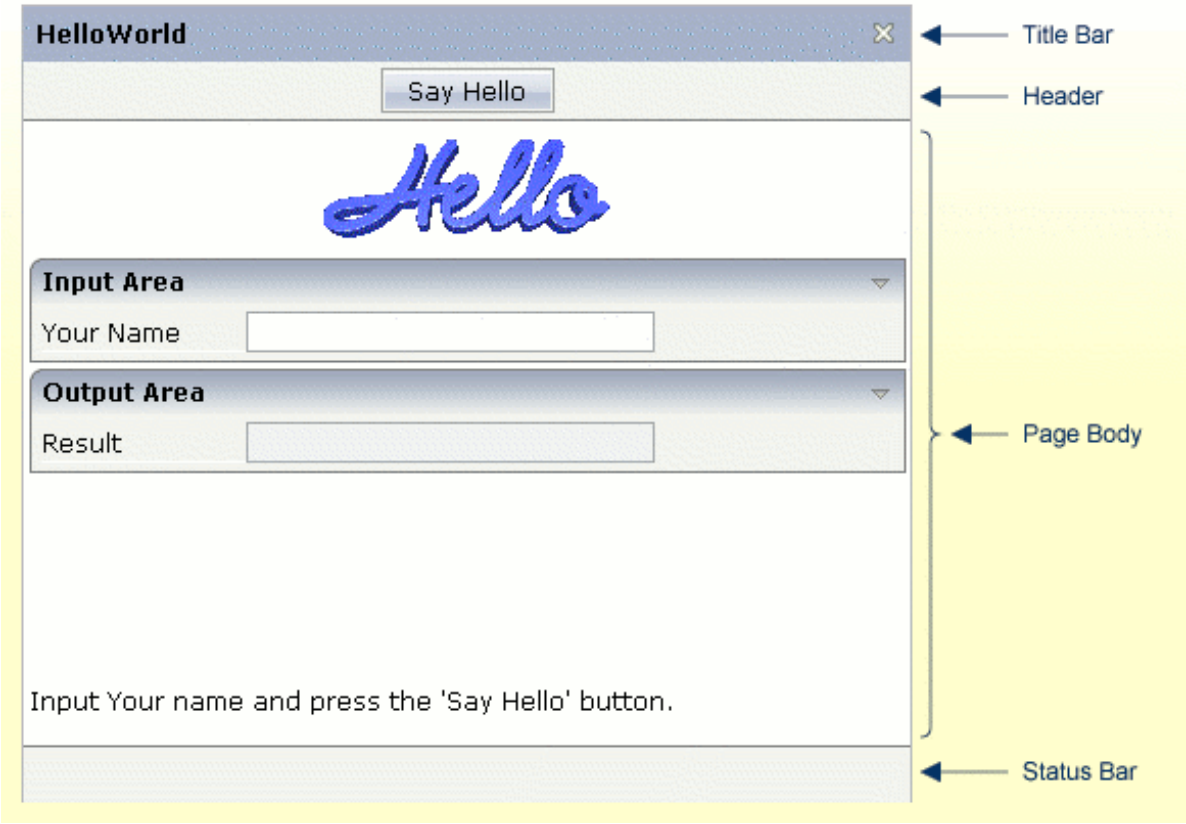
- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Typical Page Layout

The layout of a page typically contains the following elements:



This part describes these elements in more detail.

PAGE

TITLEBAR

HEADER

PAGEBODY

STATUSBAR/STATUSBARSSW

2 PAGE

The PAGE control is always the top node of a page's layout definition. The page, on the one hand, generates the visible container in which all the contained elements are placed; on the other hand, some important logical settings are defined on page level:

- The property `model` defines the name of the adapter class that is the logical counterpart on the server side.
- The property `translationreference` defines the name of the “text pool” that is used for translating literals into language-dependent texts.

Properties

Basic			
model	This is the name of the Java class that is the logical counter part of the page on server side. The name must include the full class name e.g. including the package name. Example: if you have a class DemoAdapter inside the package com.xyz.demo, the MODEL value is: com.xyz.demo.DemoAdapter. The class must be a valid adapter class i.e. it must support the interface "com.softwareag.cis.server.IModel". This is implicitly done when deriving your adapter class from "com.softwareag.cis.server.Model". The class source code may be generated by using the Code Assistant - or may be directly coded in a development environment of your choice. You may use the class "DummyAdapter" for testing your layout - before specifying your "real" class.	Obligatory	
translationreference	This is the "translation reference" that is passed to the multi language management. The "translation reference" is a logical term representing a group of textids together with their translation. If using the standard file based multi language management that comes with CIS as default then a "translation reference" represents one file containing text-ids and translations in a comma separated format. Translation information is loaded by the multi language management "per translation reference". I.e. if a page links to a certain translation reference then all the translation information that is available through this reference is loaded in one step and is also buffered.	Sometimes obligatory	

	You can set up different scenarios: either each page may address an own translation reference. E.g. if your page is named "abc.xml" then it references to "abc" - as consequence there is (per language) one abc.csv file holding translation information for this page. If you have a second page "def.xml" then you may define "def" accordingly. In this case each page is independent from the other. - On the other side you are required to translate certain "common text-ids" multiple times. If you on the other hand define one translation reference for multiple pages then you can share text-ids throughout the various pages. Please set up a strategy for using translation references when starting using the multi language management. The strategy should also include a structured way of naming text-ids. Text-ids may only be shared in an efficient way if it is clear what they stand for. E.g. you may names of buttons in the following way: "btn_save" and "btn_saveas".		
stylesheetfile	URL of a style sheet file used for control rendering. Typically the style sheet file used for control rendering is set dynamically e.g. the style depends on the user who is currently logged on. When defining the style sheet file by this property, the style sheet file is not set dynamically but defined in a fix way for this page. The style sheet file must be defined as URL, relative to the generated page. A valid value may be "../softwareag/styles/CIS_DEFAULT.css". If not using the "hard setting" of the style sheet file via this property then the style sheet is determined by the runtime in the following way: (1) The adapter object provides for a "String getStyle()" method that return the URL. You can override the default method and pass back your own URL. (2) When using the default implementation derived from com.softwareag.cis.server.Model then the getStyle() method accesses the CIS session context. You can set the session's style by calling "findCISessionContext()" in your adapter and calling "setStyle()" in the session context's object.	Optional	css
responsivestylesheetfile	URL of the responsive stylesheet file for Bootstrap. If not specified the default Bootstrap stylesheet file is used. For more information see the responsive style guide. For non responsive pages this property is ignored.	Optional	

datatablesstylesheetfile	URL of the responsive DataTables stylesheet file for rendering responsive grids. If not specified the default DataTables stylesheet file is used. For more information see the responsive style guide. For non responsive pages this property is ignored.	Optional	
addstylesheetfile	URL of an additional style sheet file. You may use this additional style sheet file in order to define more styles than are provided in the "normal" style sheet file. Typical situations are: (A) Some controls offer the possibility to render defined content by style-class definitions (e.g. inside a TEXTGRID you can dynamically define which style-class is used for a certain cell). (B) If you define own controls by using the control extension framework and if these controls require own style classes then these style classes may be provided inside the additional style sheet file. By using the additional style sheet file you are able to avoid doing manipulations to the "normal" style sheet files that come from CIS or that are generated inside the tool "Style Sheet Editor".	Optional	css
imagestopreload	Semicolon separated list of image-URLs that are directly preloaded in an invisible area of the page. If images are used inside a tree or a text grid then they are loaded by dynamically generated HTML that is placed into a corresponding area of the page. In order to optimise the loading you can preload such images by listing them in this property. The URL of the images must be relative to your generated HTML page. Example: if your page has a tree with certain node images then you may define: "images/nodeopened.gif" images/nodeclosed.gif; images/nodeendnode.gif".	Optional	
darkbackground	Normally a page background is in light colour (white if using CIS_DEFAULT style sheet). CIS style sheets also have a dark(er) grey colour to be used. If DARKBACKGROUND is set to true then the darker background colour is chosen. This property typically is used if using the SUBCISPAGE tag or ROWTABSUBPAGES tag to seamlessly integrate inner pages into darker container areas.	Optional	true false

helpid	This is the id that is passed into the help management for the page. If a user clicks F1 inside the page and if there is no specific context sensitive control help available (e.g. help for field) then the help for the page is popped up.	Optional	
visiblevalueifundefined	Several CIS controls support a VISIBLEPROP property. The VISIBLEPROP contains the binding to an adapter property that decides at runtime if a control is visible or not. This property defines how these controls behave if there is no implementation available for the property. Example: the VISIBLEPROP of a CHECKBOX is binding to a property "cbvisible" but there is not corresponding implementation "getCbvisible". If set to "true" then all controls with undefined visibility are displayed. If set to "false" then they are hidden.	Optional	true false
contextmenumethod	Name of an adapter method that is invoked if the user clicks into the page with the right mouse button and no other control (e.g. textgrid, tree,...) handled the click so far.	Optional	
immediatedisplay	Flag that indicates if the screen is visible within the initial loading phase. Default is false. When using the default you see a light HTML page showing a "just loading" image. Use property "justloadingurl" to specify a page of choice.	Optional	true false
addjavascriptlibs	Comma separated list of URLs of additional javascript libraries. Example: "../yourproject/js/yourlib.js". Used to include non-CIS javascript. Example of Usage: with the DATEINPUT control you can run own rules to convert and validate user input.	Optional	
flushmethod	Name of an adapter method that is invoked in case the page loses the focus to another CIS page.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
adapterlisteners	Semicolon separated list of classes which connect to the server side adapter processing as adapter listeners (each one supporting the interface IAdapterListener).	Optional	
openajaxsupport	Adds registration code into the page that registers globally used objects / evets etc. to the Open AHAX Hub in order to potencially synchronize the co-existance of different toolsets within one page. Only used when being familiar with OpenAJAX aspects.	Optional	true false
framebufferpriority	Priority (integer) that is used to manage the page within the CIS frame buffer. Use value "-1" to indicate that the	Optional	0

	page should not be buffered at all (typically used when having a FILEUPLOAD2 control on the page). Default is "0". Use any other integer value to indicate higher priority.		-1
centralcontextmenu	If set to 'true' then the context menu is rendered in a central frame. This central frame can be specified via the "popupdivframe" setting in cisconfig.	Optional	true false
usexmlhttprequest	By default CIS framework is using hidden frame communication (asynchronous server communication). Use this attribute in order to use "XMLHTTPRequests". Typical usage is with timer pages (to avoid seeing ongoing communication to server on browser's statusbar).	Optional	true false msie mozilla
withownborder	If set to "true" the page will be surrounded by an additional border.	Optional	true false
userinputprop	Property of the adapter which will have a value of "true" if some userinput in the page or one of its subpages has been done since the last server-roundtrip.	Optional	
Popup			
popupwidth	Each CIS page can be opened as a popup dialog. This properties define the pixel width preferred for the page. - See the property "popupheight" for more information.	Optional	100px 200px 300px 400px
popupheight	Each CIS page can be opened as a popup dialog. This property defines the pixel height preferred for the page. A popup is typically opened by calling the "openPopup"-method in your adapter code. If no further definition is done then the popup will open in the height that is defined by this value. You can also dynamically manipulate the size and position of the popup by using the Model-method "setPopupFeatures" - please read corresponding documentation inside the Java API documentation.	Optional	100px 200px 300px 400px
popupfeatures	In addition to POPUPWIDTH and POPUPHEIGHT you can control the appearance of the popup dialog in which the current page may be displayed. You define a string to maintain different feature aspects, separated by semi-colon. center:yes|no	Optional	dialogLeft: 200px dialogTop: 100px edge: sunken

	edge:sunken|raised resizable:yes|no scroll:yes|no status:yes|no (to display or hide a status bar) An example string looks as follows: "dialogLeft:100px" There is one special function built in by which you can position a popup relative to its caller's window (the dialogLeft and dialogTop definition normally refer to absolute coordinates of the screen): by specifying "dialogLeft: SCRX(100)px" you define that the position is 100 pixels right from the left top corner of the current window. - Use "dialogTop: SCRY(100)px" in the same way for vertical positioning. Please also pay attention to the methods "setPopupTitle()" and "setPopupPageFeatures()" in the com.casabac.server.Model class. By using these method you can define popup parameters in a dynamic way inside your adapter implementation.		resizable: yes status: no
Occupied			
occupiedimage	URL of the image that is displayed to indicate that the screen is just communicating to the server. This is the image that is located in the top left corner and which by default is a flashing hour glass. You can specify any image, e.g. also animated GIF files. If you want your image not to be visible in the top left corner but "somewhere" in the screen then draw an image with some transparent area on the left and above the image that you want to show.	Optional	
occupiedpixelheight	When the screen is busy, because the client is exchanging information with the server, an hour glass image is displayed at the top left corner. With this property you define the pixel height of this hour glass image.	Optional	
occupiedpixelwidth	When the screen is busy, because the client is exchanging information with the server, an hour glass image is displayed at the top left corner. With this property you define the pixel width of this hour glass image.	Optional	
Hot Keys			
hotkeys	Semicolon separated list of hot keys. A hotkey consists of a list of keys and a method name. Separate the keys by "-" and the method name again with a semicolon Example:	Optional	

	ctrl-alt-65;onCtrlAltA;13;onEnter ...defines two hot keys. Method onCtrlAltA is invoked if the user presses Ctrl-Alt-A. Method "onEnter" is called if the user presses the ENTER key. Use the popup help within the Layout Painter to input hot keys.		
Loading			
justloadingurl	URL of the page that is displayed to indicate that screen is just loading. Typically this is a light HTML page showing a loading image of choice. Use plain HTML - not a generated CIS page.	Optional	

3 TITLEBAR

The title bar is typically placed at the top of a page. The text in the title bar can either be set statically inside the layout definition, or it can be dynamically resolved by a property of the corresponding adapter.

The title bar can have a close icon (cross at the top right) and an online help icon. The close icon always calls the method `endProcess` of your adapter. This method is provided by the derived adapter class. The default implementation of the `endProcess()` method in the adapter class forces the session management to release the adapter for garbage collection.

You can overwrite the method in your adapter - but do not forget to call the superclass's method as well.

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	
withclose	In the right top corner of the titlebar there is by default a close-icon. Define "false" in this property in order to hide this icon. The close-icon calls the method "endProcess" of your adapter. "endProcess" is implemented in the class "com.softwareag.cis.server.Model" and by default ends the subsession the adapter is running in. - Override this implementation if this default implementation does not fit to your needs.	Optional	true false
align	Horizontal alignment of the text that is shown.	Optional	left center right
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:	Optional	

	(A) Define the URL relative to your page. Your page is generated directly into your project's folder. specifying "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".		
helpid	Id that is passed to the online help management. If this "helpid" is specified then a help-icon will be displayed in the right top corner. If clicking on the icon then the corresponding help will show up.	Optional	
titlestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
pixelheight	Height of the control in pixels.	Optional	1 2 3 int-value
straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".	Optional	true false
closetitle	The text that is entered here appears as tooltip on the close-icon on the right top border of the titlebar.	Optional	
closetitletextid	Multi language dependent text that displays the tooltip on the close-icon. Do not specify a CLOSETITLE if you are specifying a CLOSETITLEID.	Optional	

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
valueprop	Property of the server side adapter from which the titlebar text is dynamically derived. In situations in which the titlebar should contain some context dependent information you specify an adapter property to provide the text for the control. Do not use "name" or "textid" when using this "valueprop" property.	Optional	
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
withcloseprop	Name of adapter property that indicates if the close icon of the titlebar is visible. The server side property needs to be of type "boolean".	Optional	

4 HEADER

The header is an area in which you can place buttons, icons and menus. The area itself is grey and has a dark grey line at its bottom (if using the standard style sheet). The header is used to display buttons and icons that are valid for the whole page. Typically, it is placed directly under the title bar.

Properties

Basic			
nocellspacing	Flag that indicates if there is space between controls within the the header table. Default is FALSE.	Optional	true false
align	Horizontal alignment of the control's content. Default is "center".	Optional	left center right
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
withdistance	If set to TRUE then an additional distance will be added at the bottom of the header. Default is FALSE.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

5 PAGEBODY

The page body is the main area in which you place the body part of your layout. The body adapts its height to the current window's height, while elements such as TITLEBAR, HEADER and STATUSBAR keep a constant height. If the page body's size is too small to hold its content, you scroll through the elements that are inside the PAGEBODY.

Properties

Basic			
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Optional	auto scroll hidden
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Optional	auto scroll hidden
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	Optional	true false
pagebodystyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Padding			
horizdist	Defines if there is always a small horizontal distance kept between the border of the PAGEBODY area and its content. Set to 'false' if you want controls in the page body to directly start at the very left and to end at the very end - without any distance. Default is 'true'.	Optional	true false
paddingleft	Number of pixels which you want to keep as margin between the pagebody's border and its content. If you want that all contents inside your page body keeps a horizontal distance of 50 pixels on the left then specify: PADDINGLEFT = 50 The PADDINGLEFT and PADDINGRIGHT values are added in addition to the small horizontal distance which is added via the HORIZDIST property.	Optional	1 2 3 int-value
paddingright	Number of pixels which you want to keep as margin between the pagebody's border and its content. If you want that all contents inside your page body keeps a horizontal distance of 50 pixels on the right then specify: PADDINGRIGHT = 50 The PADDINGLEFT and PADDINGRIGHT values are added in addition to the small horizontal distance which is added via the HORIZDIST property.	Optional	1 2 3 int-value
paddingtop	Number of pixels which you want to keep as margin between the pagebody's border and its content. If you want that all contents inside your page body keeps a vertical distance of 50 pixels on the top then specify: PADDINGTOP = 50	Optional	1 2 3 int-value
paddingbottom	Number of pixels which you want to keep as margin between the pagebody's border and its content. If you want that all contents inside your page body keeps a vertical distance of 50 pixels on the bottom then specify:	Optional	1 2

	PADDINGBOTTOM = 50		3 int-value
Logon Form			
withformtag	Default value is false. If set to true all controls included in the pagebody tag will be surrounded by a form tag - only in the generated html page. That makes it possible to save or transfer forms. i.e. save username and password or a complete search form. You will also need an 'submitbutton' - please have a look at the button control.	Optional	true false

6 STATUSBAR/STATUSBARSSW

STATUSBAR

Normally, the status bar is located at the bottom of a page. It is a grey area (if using the standard style sheet) where status information can be seen. The status information is derived dynamically from properties of the adapter class. The information consists of three parts:

- Type of the status message - whether it is an error message, a warning or a success message. Depending on the type, a small icon is displayed to the left of the message.
- The status message itself - the text displayed within the status message.
- A long text for the status - text shown in a dialog when clicking on the status message.

Adapter Implementation Details

The adapter class has to support properties which provide data that are passed to the status bar. The get methods for the default property names are:

```
public String getMessageType() ...
public String getMessageShortText() ...
public String getMessageLongText() ...
```

They are already implemented by the adapter class: normally you do not have to overwrite these methods.

There is a method `outputMessage(String type, String shortText, String longText)` that can be used from the adapter class inside your adapter implementation.

Example: In the "Hello World!" application, you want to display an error message if the user clicks the **Say Hello!** button and has not yet entered a name. The `sayHello()` method looks as follows:

```
public void sayHello()
{
    if (m_name == null || m_name.trim().length()== 0)
    {
      this.outputMessage("E","Please enter a user name !"," ... space for a longer ↩
text ...");
       return;
    }
    m_result = "Hello World, " + m_name + "!";
}
```

The method `outputMessage(String type, String shortText)` does not need a long message text. In this case, the `sayHello()` method looks as follows:

```
public void sayHello()
{
    if (m_name == null || m_name.trim().length()== 0)
    {
       this.outputMessage("E","Please enter a user name !");
       return;
    }
    m_result = "Hello World, " + m_name + "!";
}
```

The screen including the error message looks as follows:

Three message types are available:

- "E" for errors.
- "W" for warnings.
- "S" for success and informational messages.

There are corresponding constants in the `Adapter` class:

- `Adapter.MT_ERROR`
- `Adapter.MT_SUCCESS`
- `Adapter.MT_WARNING`

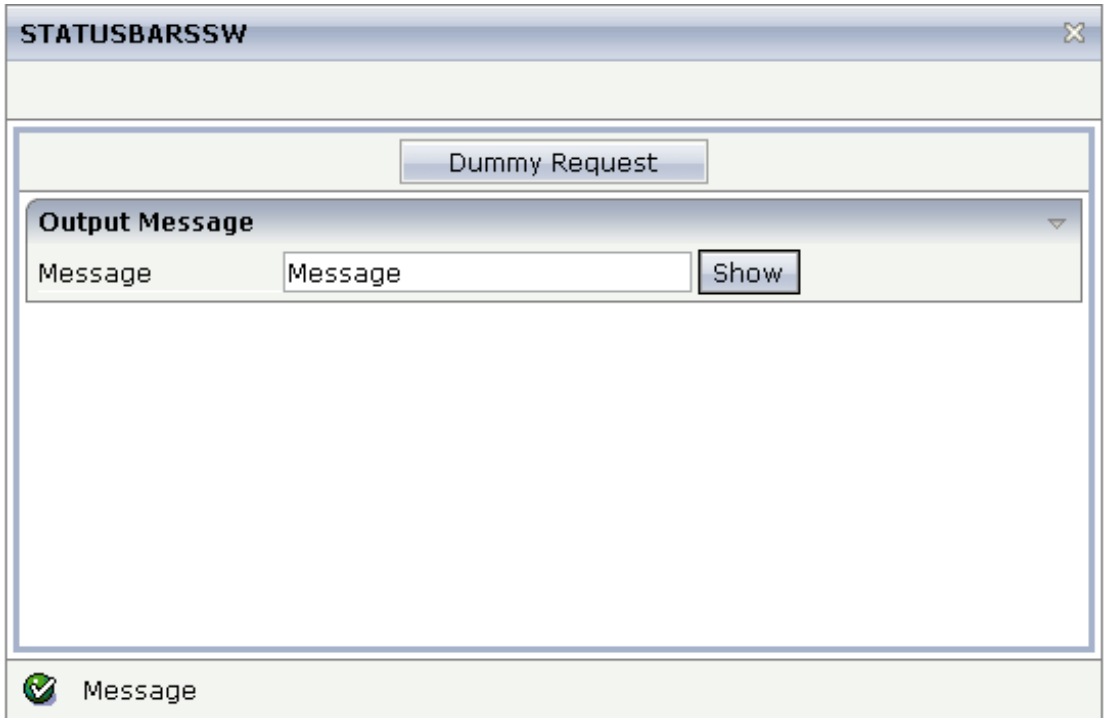
STATUSBARSSW

The STATUSBARSSW control is an extension to the normal STATUSBAR control. "SSW" in the control name stands for “subsession-wide”.

The STATUSBARSSW control allows you to output messages in the statusbar from any subsession, even from a SUBCISPAGE2 control. You place the STATUSBARSSW control once, and you can call from anywhere.

Example

In this example, the SUBCISPAGE2 control sends an `outputMessage()` to the STATUSBARSSW control of the parent page.



The XML layout definition is:

```
<pagebody takefullheight="true">
  <itr takefullwidth="true" height="100%">
   <subcispage2 subcispageprop="subcisPageInfo" width="100%" height="100%">
   </subcispage2>
  </itr>
</pagebody>
<statusbarssw>
</statusbarssw>
```

There is no need for any special coding in order to use the STATUSBARSSW control. Both controls use the same server API.

```
public void onShowMessage()
{
  outputMessage(MT_SUCCESS, m_message);
}
```

Advantage of the STATUSBARSSW Control

The following coding would be necessary with a normal STATUSBAR control:

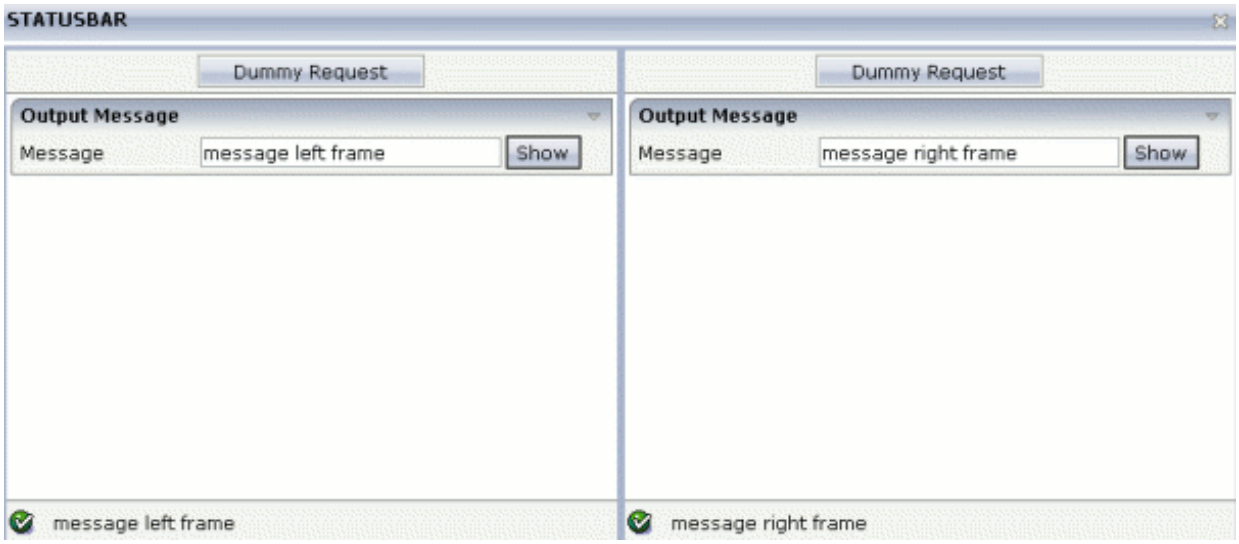
```
public void onShowMessage()  // old version
{
  // find the 'Outer Adapter'
  StatusbarSSWAdapter outer =
            (StatusbarSSWAdapter)findAdapter(StatusbarSSWAdapter.class);
  // output the message
  outer.outputMessage(MT_SUCCESS, m_message);
  // and refresh the parent page.
  refreshParent();
}
```

This example shows only the code which is required for a page which is nested once. A lot of additional code would be required for further nested pages. When using the STATUSBARSSW control, you need not worry about nesting.

When to use the STATUSBARSSW Control

If you have many pages (maybe with several included SUBCISPAGE2 controls) and you want to have a single status bar for all of your messages, you use the STAUSBARSSW control.

If you have only a few pages (for example, 2 pages in a frameset) and you want to output a different message for each page, you have to use the normal STATUSBAR control. For example:



STATUSBAR Properties

Basic			
typeprop	Name of the adapter property holding the information about the type of the status message. The type defines the image that is rendered at the beginning of the message. Currently there are 3 supported values that can be passed back from the property: E for error, W for warning, S for success. The default property name is messageType provided by the Model-class, from which you derive your adapter class. Please pay attention: changing this property means that you also have to override the "outputMessage(...)" methods inside your adapter accordingly.	Optional	
shorttextprop	Name of the adapter property providing the message text that is visible inside the status bar. The default property name is messageShortText and is provided by the Model-class. Please pay attention: changing this property means that you also have to override the "outputMessage(...)" methods inside your adapter accordingly.	Optional	
longtextprop	Name of the adapter property providing the long message text. The long text pops up if clicking onto the short text message. The default property name is messageLongText and is provided by the Model-class. Please pay attention: changing this property means that you also have to override the "outputMessage(...)" methods inside your adapter accordingly.	Optional	

straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".	Optional	true false
resetbefore	If set to TRUE, the control is reset before a server roundtrip is done.	Optional	true false
withdistance	If set to TRUE then an additional distance will be added at the top of the statusbar. Default is FALSE:	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
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	

STATUSBARSSW Properties

Basic			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
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	

II Working with Containers

Containers are areas on your screen that can hold controls (such as fields, labels, etc.) or other container(s). Containers are the preferred way to structure elements inside your page body.

You already saw the ROWAREA container in the "Hello World!" example.

The information provided in this part is organized under the following headings:

7 Positioning of Controls inside a Container

Containers internally build an HTML table in which you place rows. Inside each row you place the controls - or again container(s).

Row Types - TR and ITR

There are two types of rows:

- The TR row is a normal table row. If you place more table rows - one under the other - inside one container, the columns inside the table row are all synchronized. See the example below in order to understand what “synchronized” means.

  Since controls are placed into columns, all controls are positioned in a synchronized way.
- The ITR row is a special table row. If you place more ITR table rows - one under the other - inside one container, each row has an independent set of columns; i.e. columns are not synchronized.

Have a look at the following XML layout description:

```
<rowarea name="With TR">
    <tr>
        <label name="First Name" width="100">
        </label>
        <field valueprop="fname" width="200">
        </field>
    </tr>
    <tr>
        <label name="Last Name" width="200">
        </label>
        <field valueprop="lname" width="200">
        </field>
    </tr>
</rowarea>
<rowarea name="With ITR">
    <itr takefullwidth="true">
        <label name="First Name" width="100px">
        </label>
        <field valueprop="fname" width="200">
        </field>
    </itr>
    <itr takefullwidth="true">
        <label name="Last Name" width="200">
        </label>
        <field valueprop="lname" width="200" length="20">
        </field>
    </itr>
</rowarea>
```

Note that each control (label, button, fields, etc.) is placed into one column of its own. If you have many controls inside one row - and have several rows one below the other - synchronized columns (using TR rows) sometimes cause funny results.

What is better, TR or ITR? Of course, it depends. The recommendation is:

- Use ITR as default. Using ITR, each row is defined independently from other rows that are positioned in the same container. You can change the number of controls (i.e. you internally change the number of managed columns) in one row without interdependencies to other rows.
- Only use TR if you really want to synchronize columns. A typical area of usage is inside the grid management (ROWTABLEAREA2 control): in a grid you explicitly desire to have synchronized columns inside the grid's table.

Some More Details on ITR

There are two ROWAREA containers. The first one uses TR rows, the second one uses ITR rows. The label for **First Name** has a width of 100 pixels, the label for **Last Name** has a width of 200 pixels. Now look at the result:

Inside the TR rows, all columns are synchronized - while in the ITR rows, each row is individually arranged.

How does the ITR control work internally? For each row, an individual table is opened with one row. Example: you define the following area in the XML layout definition:

```
<area>
  <itr>
    ...
    ...
  </itr>
  <itr>
    ...
    ...
  </itr>
</area>
```

The generated HTML looks like this:

```
<table>
  <tr>
    <td colspan="100">
      <table>
        <tr>
          ...
          ...
        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td colspan="100">
      <table>
        <tr>
          ...
          ...
        </tr>
      </table>
    </td>
  </tr>
</table>
```

Inside each row there is a table definition of its own, holding exactly one row.

You can define a `takefullwidth` property with the ITR definition, defining the width of the internal table of an ITR tag. If the `takefullwidth` property is set to "true", this means that the internal table that is kept per row is internally opened to use 100% of the available width. Without any definition, the table will be as big as it is required by its content.

ITR in Google Chrome and Edge Chromium

When using pixel sizing for controls in ITRs you may see rounding issues when zooming the page in Google Chrome and Edge Chromium. See the example "Inline rendering" in the NaturalAjax-Demos.

In case your ITR only contains the following controls: FIELD, LABEL, HDIST, ICON, BUTTON and/or XCIDATADEF, you can set the inline property to true. This will force the browser to use a different rendering style, which avoids the rounding issues while zooming. Instead of setting the inline property directly in the ITR you can specify this inline rendering for the whole page or the whole application.

To render this kind of ITRs of the whole page inline, set

```
<natpage itrinlinedisplay='true' ...
```

To render this kind of ITRs in the whole application inline, in *cisconfig.xml*, set

```
<cisconfig itrinlinedisplay='true' ... ↩
```

TR Properties

Basic			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). Please note: the row content may overrule this setting. The height setting "100px" of an embedded textbox will beat a row height of "50px". (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
withalterbackground	Flag that indicates if the grid line shows alternating background color (like rows within a textgrids). Default is false. Please note: controls inside the row must have transparent background. In case of the FIELD control simply set property TRANSPARENTBACKGROUND to true.	Optional	true false
trstyle	CSS style definition that is directly passed into this control.	Optional	background-color: #FF0000 color: #0000FF

	With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		font-weight: bold
trstyleprop	Name of adapter property that dynamically provides explicit style information for the control.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

ITR Properties

Basic			
takefullwidth	If set to "true" then the control takes all available horizontal width as its width. If set to "false" then the control does not have a predefined width but grows with its content.	Optional	true false
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does	Optional	100 150 200 250 300 250 400 50% 100%

	not specify a width then the rendering result may not represent what you expect.		
align	Alignment of the content of the ITR row. Background: the ITR as independent table row renders a table into its content area. Inside this table a row is opened in which the controls are placed. This table normally is starting on the left of the ITR row. With this ALIGN property you can explicitly define the alignment of the table.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.	Optional	true false
inline	Only set this property to true if you see rounding issues when zooming your page in Google Chrome or Edge Chromium browser. The property will force the browser to use a different rendering style for this itr. Use this property only if your ITR only contains the following controls: FIELD, LABEL, HDIST, ICON, BUTTON and/or XCIDATADEF and you are using pixel sizing.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Visibility			

visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
Appearance			
itrstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
itrclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Binding			
itrstyleprop	Name of server side property/data element that directly controls the style of the control.	Optional	

8 Defining the Width of Controls inside a Container

As mentioned in the previous section, each control is automatically embedded into a column. Consequently, the width of the control is, on the one hand, determined by the size of the control itself - on the other hand, the column is part of a table row and also follows the table row's sizing.

Controlling the Width of Controls

Every control that allows width sizing offers a corresponding `width` property. In this property, you put either an absolute pixel value (`width="100"`) or a percentage value (`width="50%"`). The rendering follows the strategy:

- If the width of a control is specified as a pixel value, the width is fixed: if the browser screen is too small to display all controls, the controls will not be reduced but keep their pixel size. Depending on your settings in the PAGEBODY tag (`hscroll` property), the displayed elements will be cut off or will be accessible by a scroll bar.
- If the width of a control is defined as a percentage value (`width="50%"`), HTML renders the control accordingly. If the screen is too small to show all controls, the browser will try to reduce elements according to the table rendering rules.

If you define the width of a control as a percentage value, the width relates to

- the width of the area in case of using TR rows, or to
- the width definition of the ITR row if using ITR rows. This width definition can either be absolute or percentage-based.

The following example shows a page in which controls hold percentages values for the width:

```
<itr takefullwidth="true">
    <label name="Factor1" width="20%">
    </label>
    <field valueprop="factor1" width="80%">
    </field>
</itr>
<itr takefullwidth="true">
    <label name="Factor2" width="20%">
    </label>
    <field valueprop="factor2" width="60%">
    </field>
    <hdist width="20%">
    </hdist>
```

The HTML page looks as follows - the size of the controls changes according to their percentage definition:

A similar screen is now built using absolutely defined pixel sizes:

```
<itr takefullwidth="false">
    <label name="Factor1" width="100">
    </label>
    <field valueprop="factor1" width="200">
    </field>
</itr>
<itr takefullwidth="true">
    <label name="Factor2" width="100">
    </label>
    <field valueprop="factor2" width="150">
    </field>
</itr>
```

In the ITR definition, there is no `width` specification - therefore, the controls will occupy exactly the space they require. The result looks as follows - the size of the controls will not change when changing the screen size:

Pay attention to what was said previously: Controls are placed into columns; columns are placed into table rows; and table rows are placed into containers. If you place a control into a row and define this control to have a width of 100%, then the elements “above” have to take care of providing the space to which the control relates its "100%". More concrete: If you place a FIELD control with a width of 100% into an ITR row that does not provide for a width of 100% itself (using the property `takefullwidth`), then the result will be a minimum-width field (100% of nothing).

Pixel sizing represents a bottom-up sizing approach: a control defines its width - all the other controls around (e.g. the container in which the control is placed) have as a consequence to adapt to the control's size: if the control is defined to occupy more space, then the container has to follow and provide for the space.

Percentage sizing represents a top-down sizing approach: the inner control tells how many percentages of the space that is granted from the outer control is occupied. As a consequence the outer control needs to define its size properly. Either the outer control itself defines a pixel size or it itself defines a percentage size - thus passing the responsibility to the next higher level. This might end up in a cascading definition of “percentage sizing” - up to the PAGEBODY control, which is the outer-most container of a page.

There are four commonly used properties for sizing:

- `width/height` - this is the quite obvious definition as explained in this section.
- `takefullwidth/takefullheight` - this is an equivalent to `width="100%"` and `height="100%"`.

HDIST and VDIST Controls

HDIST means “horizontal distance”. VDIST means “vertical distance”.

HDIST Control

The HDIST control represents a distance to be placed between controls. The distance itself holds a certain width that again can either be a pixel width or a percentage width.

The following example shows a table row into which a town and a zip code is put:

Between the two FIELD controls, you see a small distance that separates the fields from one another. The corresponding XML layout definition is:

```
<rowarea name="HDIST Example">
    <itr>
        <label name="Zip Code / Town" width="120">
        </label>
        <field valueprop="zipcode" width="80">
        </field>
        <hdist width="5">
        </hdist>
        <field valueprop="town" width="200">
        </field>
    </itr>
</rowarea>
```

The HDIST control is also very useful for percentage-based sizing of widths. If you want a control to occupy 50% of the available width, you have to “fill the gap” in the following way:

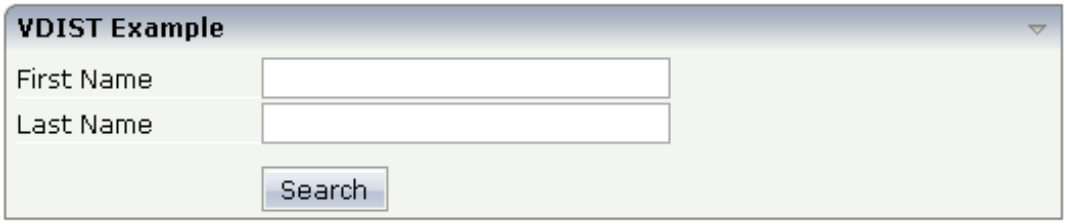
The corresponding XML layout definition is:

```
<rowarea name="HDIST Example">
    <itr height="100%">
        <label name="First Name" width="120">
        </label>
        <field valueprop="fname" width="50%">
        </field>
        <hdist width="50%">
        </hdist>
    </itr>
</rowarea>
```

Pay attention: when using percentage sizing, then you should take care of filling the "100%" by the controls inside the row. Otherwise, the browser will distribute the remaining space to its columns - i.e. the controls will not be positioned the way you expect.

VDIST Control

The VDIST control is the counterpart of the HDIST control - in vertical direction. The following example shows a scenario in which the line containing the BUTTON control keeps a vertical distance of 10 pixels from the lines containing the FIELD controls:



The layout definition is:

```
<rowarea name="VDIST Example">
    <itr height="100%">
        <label name="First Name" width="120">
        </label>
        <field valueprop="fname" width="200">
        </field>
    </itr>
    <itr height="100%">
        <label name="Last Name" width="120">
        </label>
        <field valueprop="lname" width="200">
        </field>
    </itr>
    <vdist height="10">
    </vdist>
    <itr>
        <hdist width="120">
        </hdist>
```

```
        <button name="Search" method="onSearch">
        </button>
    </itr>
</rowarea>
```

Note that an HDIST control is used in the line containing the BUTTON control to align the button to the fields.

HDIST Properties

Basic			
width	Width of the HDIST control, either in pixels or as percentage value. If no width is defined then a default width of 2 pixels is assigned.	Optional	100 120 140 160 180 200 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	

VDIST Properties

Basic			
height	Height of the VDIST control, either in pixels or as percentage value. If no width is defined then a default width of 3 pixels is assigned.	Optional	100 150 200 250 300 250 400 50% 100%
backgroundstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

rowspan and colspan Definitions

Each control has a `colspan` and `rowspan` property that is "1" by default. This definition is directly transferred to the column definition that is placed around the control.

Example:

```
<tr>
  <control colspan="2">
  </control>
</tr>
```

If you specify the above definition, the created HTML code looks like this:

```
<tr>
  <td colspan="2" rowspan="1">
    ... control-specific HTML code ...
  </td>
</tr>
```

The usage of `rowspan` and `colspan` only makes sense in scenarios in which you define multiple rows inside one container and if you use TR rows at the same time. You do not have to pay attention to them if working in ITR rows.

Again: first check if the TR way of arranging controls is really the best approach - compared to the ITR approach. Using TR means you have to “fight” with `colspan` and `rowspan` definitions in order to properly lay out your controls. With ITR, each row is independently defined from its neighbor rows.

CELLSPAN Control

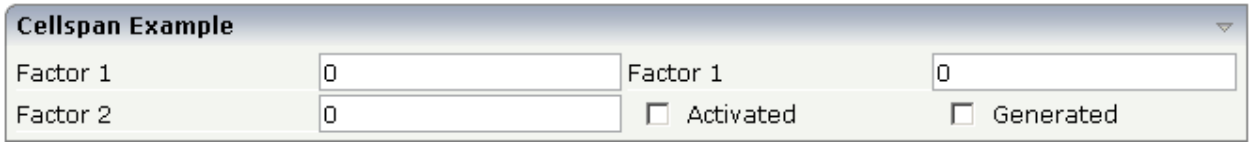
Inside one row, you can place controls or nested containers. Containers again allow you to specify new rows inside the container.

There is a special control, the CELLSPAN control. With the CELLSPAN control, you can quickly define one cell inside a row of a container to place other controls. The CELLSPAN control has a `width` property to specify the width of its inner content.

Have a look at the following example:

```
<rowarea name="Cellspan Example">
    <tr>
        <label name="Factor 1" width="25%">
        </label>
        <field valueprop="factor1" width="25%">
        </field>
        <hdist></hdist>
        <cellspan width="50%">
            <label name="Factor 1" width="50%">
            </label>
            <field valueprop="factor1" width="50%">
            </field>
        </cellspan>
    </tr>
    <tr>
        <label name="Factor 2" width="25%">
        </label>
        <field valueprop="factor2" width="25%">
        </field>
        <hdist></hdist>
        <cellspan width="50%">
            <checkbox valueprop="activated" width="10%">
            </checkbox>
            <label name="Activated" width="40%" asplaintext="true">
            </label>
            <checkbox valueprop="generated" width="10%">
            </checkbox>
            <label name="Generated" width="40%" asplaintext="true">
           </label>
        </cellspan>
    </tr>
</rowarea>
```

Each TR row contains one CELLSPAN definition with a width of 50%. The inner content of the CELLSPAN definitions is completely different between the rows:



You could add controls to the CELLSPAN definition in the first row without any implications inside the second row. The CELLSPAN control internally operates similar to the ITR control: it builds a table on its own and decouples its content from the surrounding table rendering.

CELLSPAN Properties

Basic			
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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
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 language management - representing the tooltip text that is used for the control.	Optional	
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 synchronized 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
colspanprop	Colspan property name.	Optional	
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
cellstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

backgroundclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	

Rules for Positioning Controls inside Containers

This is a collection of rules you should consider when positioning controls inside containers:

- Make up your mind where to use relative percentage values or absolute pixel definitions.
- Do not mix percentage and pixel values inside one container.
- Internally, Application Designer controls are mapped to the HTML tags `TABLE`, `TR` and `TD`. When developing, you should have in mind the normal HTML table management.
- Structure your container not as one big container holding one complex table, each row holding a lot of controls. Instead, use the possibility to define nested containers or CELLSPAN controls in order to structure your layout.

9 Vertical Sizing of Containers and Controls

Nearly all controls which can be sized offer vertical sizing by a corresponding `height` property. You can set the value of this property either as a pixel value or as a percentage value.

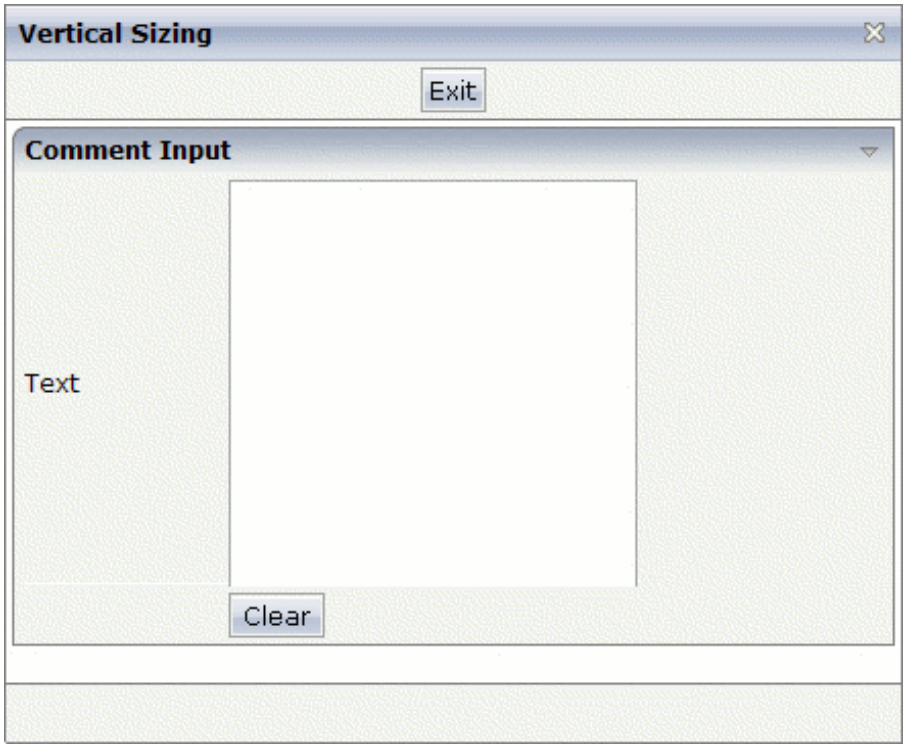
Vertical Pixel Sizing

This is the default. Controls either occupy their standard height or the height is explicitly defined in pixels. The whole page is sized from the bottom to the top.

Look at the following example:

```
<pagebody>
    <rowarea name="Comment Input">
        <itr>
            <label name="Text" width="100">
            </label>
            <text valueprop="comment" width="200" height="200">
            </text>
        </itr>
        <vdist>
        </vdist>
        <itr>
            <hdist width="100">
            </hdist>
            <button name="Clear" method="onClear">
            </button>
        </itr>
    </rowarea>
</pagebody>
```

The corresponding screen looks as follows:



The vertical size of the ROWAREA is exactly as big as required by its content. The TEXT control is defined to be 200 pixels high.

Vertical Percentage Sizing

Use the same example, but this time the size of the TEXT control should be as big as possible - depending on the size of the browser window. It should take the full available height.

The XML layout definition looks as follows:

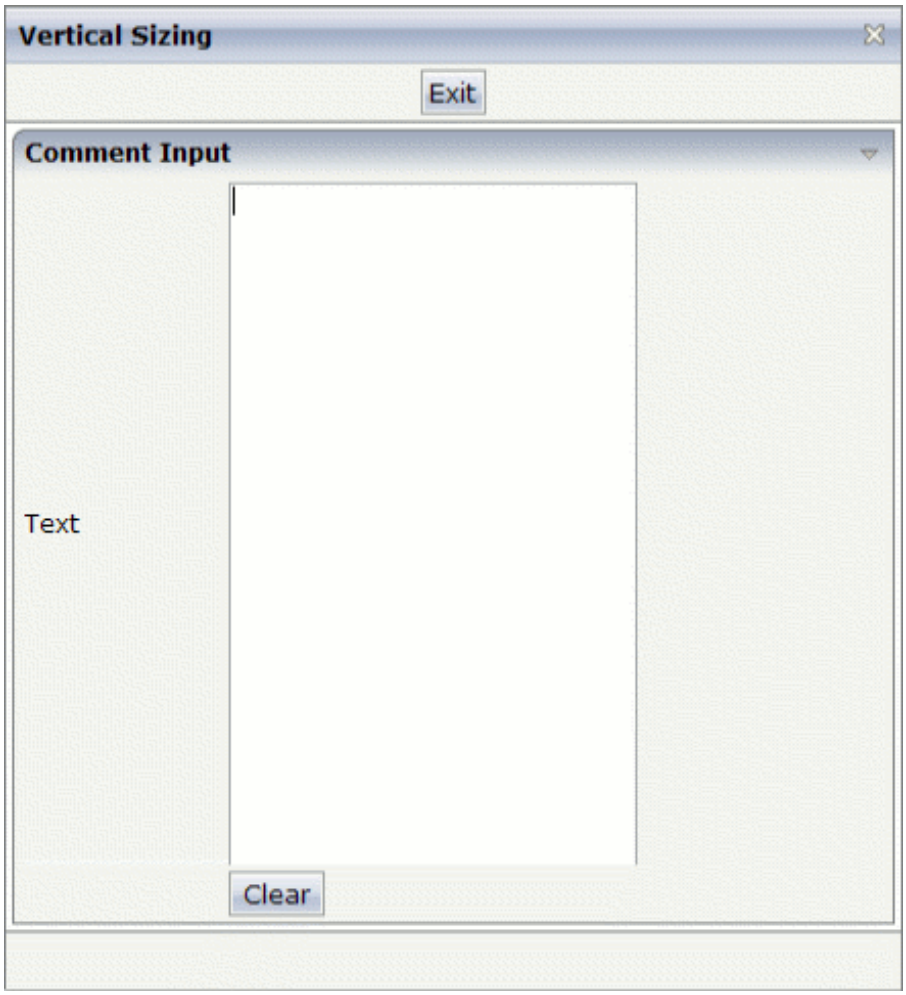
```
<pagebody takefullheight="true">
    <rowarea name="Comment Input" height="100%">
        <itr height="100%">
            <label name="Text" width="100">
            </label>
            <text valueprop="comment" width="200" height="100%">
            </text>
        </itr>
        <vdist>
        </vdist>
        <itr>
            <hdist width="100">
            </hdist>
            <button name="Clear" method="onClear">
```

```
            </button>
        </itr>
    </rowarea>
    <vdist>
    </vdist>
</pagebody>
```

The TEXT control now occupies a height of 100%. However, the definition of the whole size of the page is passed down from the PAGEBODY to the control:

- In the PAGEBODY, the property `takefullheight` is set to "true". This means that the content of the page body gets passed 100% of the available height.
- On the next level, the ITR row - in which the TEXT control is placed - is defined to have a height of "100%". This means it tries to grab as much height as possible. On the same level, there is also a VDIST (vertical distance) control and another ITR row - with no height defined. This means that these controls get as much height as they require due to their content - but the whole remaining vertical space is assigned to the first ITR row with the HEIGHT of "100%".

The result page looks as follows:



By changing the size of the browser window, the height of the whole control arrangement will follow accordingly.

You see that sizing by percentage values means that you have to think from top to bottom - just the opposite direction as you think with pixel values. This is nothing new for you if you are used to work with normal HTML tables - in fact, everything that is done below the diverse container controls is done by table rendering.

Conclusion: The example shows you that the `height` property of controls can be defined as a percentage value - but needs an outside reference to depend on. Some of the controls, such as the PAGEBODY, do not offer explicitly a `height` property but only a property `takefullheight` that can be set to "true". This is equivalent to a definition of `HEIGHT="100%"`.

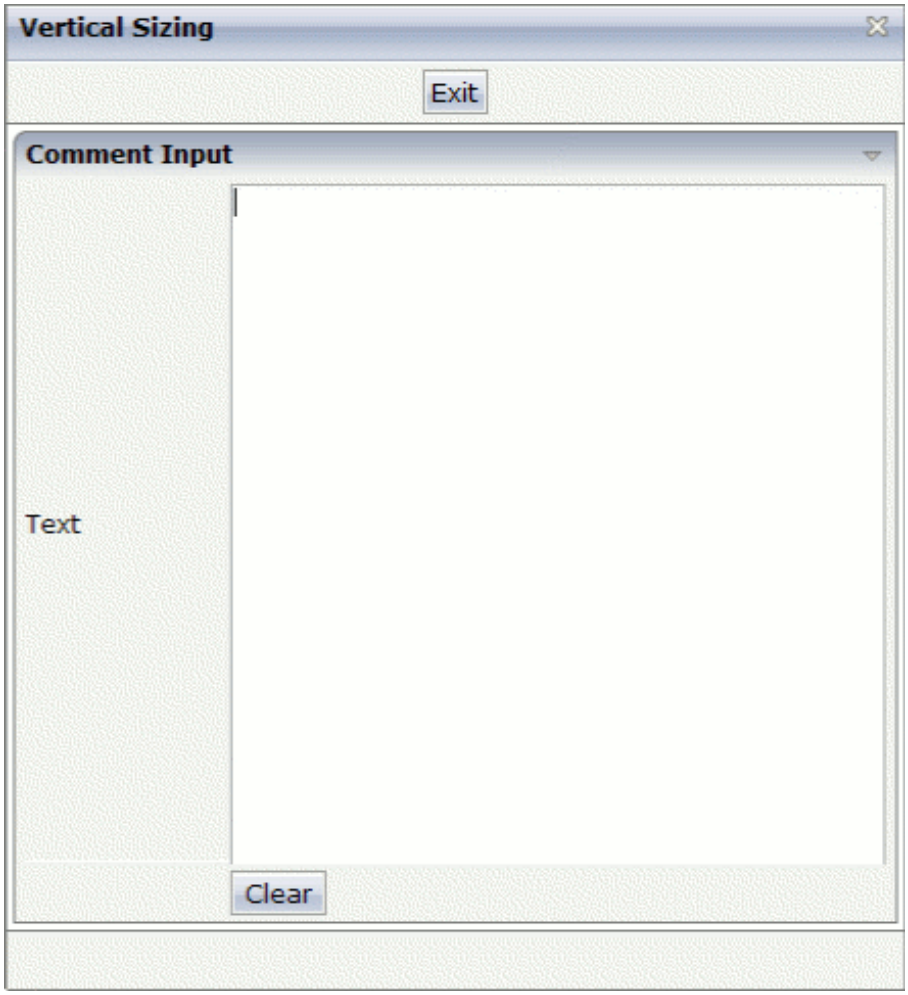
Finishing the Example

This has nothing to do with vertical sizing, but with horizontal sizing. We cannot finish the example without having changed it also in a way that it occupies the full available horizontal width. The layout definition now looks as follows:

```
<pagebody takefullheight="true">
    <rowarea name="Comment Input" height="100%">
        <itr takefullwidth="true" height="100%">
            <label name="Text" width="100">
            </label>
            <text valueprop="comment" width="100%" height="100%">
            </text>
        </itr>
        <vdist>
        </vdist>
        <itr>
            <hdist width="100">
            </hdist>
            <button name="Clear" method="onClear">
            </button>
        </itr>
    </rowarea>
    <vdist>
    </vdist>
</pagebody>
```

The `width` property of the TEXT control is set to "100%". Similar to the vertical height management, the available width is passed from the ITR row definition above - which occupies 100% of the available width inside the ROWAREA. The ROWAREA always occupies the whole available width - it does not require an explicit width definition.

The result is now:


Vertical Sizing
Exit
Comment Input
Text
Clear

10 Overview of Different Containers

Different Kind of Containers

Currently, there are the following types of containers:

- **ROWAREA and COLAREA**
  These are containers holding a title. The graphic area represented by the container is surrounded by a border. The content of the area container can be reduced by clicking on the title - and resized by clicking again on the title.
- **ROWTABAREA and COLTABAREA**
  These are containers holding different pages (TABPAGE elements) which can be toggled.
- **ROWTABLE0 and COLTABLE0**
  These are containers you do not see; i.e. a container does not have any borders or any special coloring. Use it just for arranging elements inside the container.
- **ROWDYNAVIS and COLDYNAVIS**
  This is a container that is the same as the ROWTABLE0 or COLTABLE0 container but with an additional feature: You can control the visibility of the whole container dynamically by an adapter property. Use this container if you want to display or hide a certain area of your screen depending on some business logic.

  A typical example is an address management: the user enters an address located in the United States. Therefore, an additional area has to appear in which the user enters the state information. For other countries, this area is not required and should not be visible.

Row Containers

The containers have a row implementation and a column implementation.

Row containers occupy the whole available width they can obtain. They are placed directly in other containers. You can place several row containers inside one container. Therefore, they are arranged one below the other.

Example:

```
<pagebody>
    <rowarea name="Area 1">
    </rowarea>
    <rowarea name="Area 2">
    </rowarea>
    <rowarea name="Area 3">
    </rowarea>
</pagebody>
```

The above XML layout produces the following HTML page:

Column Containers

Column containers are placed inside rows, i.e. into TR rows or ITR rows. You can place several column containers inside one row. Therefore, they are arranged in a way that one column container follows the other horizontally.

Example:

```
<pagebody>
    <itr width="100%">
        <colarea name="Area 1" width="33%">
        </colarea>
        <hdist>
        </hdist>
        <colarea name="Area 2" width="33%">
        </colarea>
        <hdist>
        </hdist>
        <colarea name="Area 3" width="33%">
        </colarea>
    </itr>
</pagebody>
```

The above XML layout produces the following HTML page:

With column containers, you have to specify the width (either as a pixel value or as a percentage value) of the container. Note that - if using percentage widths - you have to place them into an ITR row that itself occupies the whole available width (`itr width="100%"`).

Row and Column Containers in Combination

It is possible to use row and column containers in combination. The following example combines the two examples shown above.

```
<pagebody>
    <rowarea name="Area1">
    </rowarea>
    <rowarea name="Area 2">
    </rowarea>
    <rowarea name="Area 3">
    </rowarea>
    <itr width="100%">
        <colarea name="Area 1" width="33%">
        </colarea>
        <hdist>
        </hdist>
        <colarea name="Area 2" width="33%">
        </colarea>
        <hdist>
        </hdist>
        <colarea name="Area 3" width="33%">
        </colarea>
    </itr>
</pagebody>
```

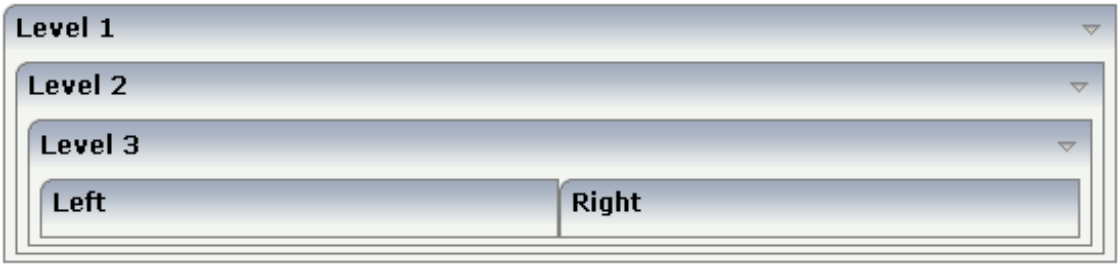
The HTML page looks as follows:

Nesting Containers

It is possible to nest containers - one into another - in any way. Example:

```
<pagebody>
    <rowarea name="Level 1">
        <rowarea name="Level 2">
            <rowarea name="Level 3">
                <itr width="100%">
                    <colarea name="Left" width="50%">
                    </colarea>
                    <hdist>
                    </hdist>
                    <colarea name="Right" width="50%">
                    </colarea>
                </itr>
            </rowarea>
        </rowarea>
    </rowarea>
</pagebody>
```

The above XML code produces the following HTML page:



11 ROWAREA and COLAREA

The ROWAREA or COLAREA container represents an area surrounded by a border and which may have a title text. By clicking on the title of such a control, the inner content is hidden (the ROWAREA or COLAREA is “folded”).

ROWAREA 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	
nameprop	Name of adapter property that provides as value the text that is shown inside the control.	Optional	
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Visibility			
foldable	The "folding"-function that is available by clicking on the title of the area can be switched off ("false"). "True" is the default.	Optional	true false
foldableprop	Name of adapter property that dynamically controls whether clicking on the title of the area will fold/unfoald this area. Valid values provided by the adapter property are TRUE (=foldable) and FALSE(=not foldable).	Optional	
foldedprop	Name of adapter property that controls whether the content of the ROWAREA is folded (true) or displayed (false). By using this property you can dynamically control the "folded"-status of the control by your adapter object.	Optional	
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You	Optional	screen server

	use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.		
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Appearance			
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. specifying "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	
imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
withtoppadding	The control by default renders some blank vertical space (2 pixels) on top of its area. Reason: if you vertically arrange one ROW/COLAREA after the other then automatically some distance is put between. By specifying "false" you can avoid this behaviour. "	Optional	true false
withleftborder	The control normally renders a black border around its area. With the properties WITHLEFTBORDER, WITHRIGHTBORDER and WITHBOTTOMBORDER you can avoid this. Reason behind: sometimes you want a ROWAREA/COLAREA to be used as	Optional	true false

	"neighbour" of other ROWAERA/COLAREA controls. In this case one of the "neighbours" has to avoid the rendering of border lines - otherwise two border lines will be rendered.		
withtopborder	See description of WITHLEFTBORDER property.	Optional	true false
withrightborder	See description of WITHLEFTBORDER property.	Optional	true false
withbottomborder	See description of WITHLEFTBORDER property.	Optional	true false
paddingleft	Number of pixels between the left border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
paddingright	Number of pixels between the right border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
areastyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

contenttablestyle	CSS style definition that is applied to the content part of the ROWAREA control.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
notabstop	The title of the area by default can be used by the user to hide/show the area's content. In order to also reach this title with the tab-key is part of the normal tab-sequence of a page. Set this property to "true" if you do not want to make the title reachable by tab-key. As consequence hiding/showing will only be available by mouse-clicking on the title.	Optional	true false
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.	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
withcontenttoppadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false
withcontentbottompadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false
withfadedtoggling	The animation of the controls can be switched off! Please take a look in your cisconfig.xml file. Set animatecontrols="true" (default) if you generally want to animate all of your controls. The rowarea control has a seperate switch (withfadedtoggling = true/false) to (de)activate the 'FadedToggling' animation especially for this single rowarea control. Notice: Entering true or false into the withfadedtoggling attribute overwrites the general animatecontrols setting !	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	
titlerowontop	Default value is 'true'. If set to 'false' the titlerow is rendered at the bottom of this area.	Optional	true false
toggleimgtitle	A text that is displayed as tooltip of the toggle image.	Optional	

toggleimgtitletextid	Multi language dependent text that is displayed as tooltip of the toggle image. Do not specify a "toogleimagetitle" inside the control if specifying a "toggleimagetextid".	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

COLAREA Properties

The properties of COLAREA are very similar to those of ROWAREA.

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	
nameprop	Name of adapter property that provides as value the text that is shown inside the 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").	Sometimes obligatory	100 120 140 160 180 200 50%

	(C) Percentage sizing: input a percentage 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.		100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	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. specifying "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	

imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.	Optional	true false
withleftborder	The control normally renders a black border around its area. With the properties WITHLEFTBORDER, WITHRIGHTBORDER and WITHBOTTOMBORDER you can avoid this. Reason behind: sometimes you want a ROWAREA/COLAREA to be used as "neighbour" of other ROWAERA/COLAREA controls. In this case one of the "neighbours" has to avoid the rendering of border lines - otherwise two border lines will be rendered.	Optional	true false
withtopborder	See description of WITHLEFTBORDER property.	Optional	true false

withrightborder	See description of WITHLEFTBORDER property.	Optional	true false
withbottomborder	See description of WITHLEFTBORDER property.	Optional	true false
paddingleft	Number of pixels between the left border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
paddingright	Number of pixels between the right border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
areastyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
contenttablestyle	CSS style that is applied to the content are of the COLAREA control.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

withcontenttoppadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false
withcontentbottompadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false
titlerowontop	Default value is 'true'. If set to 'false' the titlerow is rendered at the bottom of this area.	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	
withtoppadding	The control by default renders some blank vertical space (2 pixels) on top of its area. Reason: if you vertically arrange one ROW/COLAREA after the other then automatically some distance is put between. By specifying "false" you can avoid this behaviour. "	Optional	true false
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

12 ROWAREAWITHHEADER

This container represents an area surrounded by a border which may have a title text. By clicking on the title, the inner content is hidden (the container is “folded”). You can place icons (**ICON**, **ICONLIST**) into the header line (ROWAREAHEADER). Other content is placed into the ROWAREABODY container.

Simple Example

```
<rowareawithheader>
    <rowareaheader name="Note">
        <hdist width="20">
        </hdist>
        <icon image="../HTMLBasedGUI/images/cut.gif" method="onCut">
        </icon>
        <hdist width="6">
        </hdist>
        <icon image="../HTMLBasedGUI/images/copy.gif" method="onCopy">
        </icon>
        <hdist width="6">
        </hdist>
        <icon image="../HTMLBasedGUI/images/paste.gif" method="onPaste">
        </icon>
    </rowareaheader>
    <rowareabody>
        <itr takefullwidth="true">
            <text valueprop="text" width="100%" rows="5">
            </text>
        </itr>
    </rowareabody>
</rowareawithheader>
```

The above XML layout produces a page which looks as follows:

There are three icons within the header line (ROWAREAHEADER). The text box is placed into the body container (ROWAREABODY).

The adapter program looks as follows:

```
// property >text<
String m_text;
public String getText() { return m_text; }
public void setText(String value) { m_text = value; }

/** Method is called when clicking the Copy icon */
public void onCopy()
{
   outputMessage(MT_SUCCESS, "Copy...");
}

/** Method is called when clicking the Cut icon */
public void onCut()
{
    outputMessage(MT_SUCCESS, "Cut...");
}

/** Method is called when clicking the Paste icon */
public void onPaste()
{
    outputMessage(MT_SUCCESS, "Paste...");
}
```

Right-to-left (RTL) Mode

To properly support right-to-left (RTL) mode, it is required to set a foldableprop for correct display of right-to-left content.

For example: `<rowareawithheader foldableprop="myfoldableprop">`.

ROWAREAWITHHEADER Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20").	Optional	100 150 200 250 300 250

	(C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Visibility			
foldable	The "folding"-function that is available by clicking on the title of the area can be switched off ("false"). "True" is the default.	Optional	true false
foldableprop	Name of adapter property that dynamically controls whether clicking on the title of the area will fold/unfoald this area. Valid values provided by the adapter property are TRUE (=foldable) and FALSE(=not foldable).	Optional	
foldedprop	Name of adapter property that controls whether the content of the ROWAREA is folded (true) or displayed (false). By using this property you can dynamically control the "folded"-status of the control by your adapter object.	Optional	
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server

flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Appearance			
height	(already explained above)		
withleftborder	The control normally renders a black border around its area. With the properties WITHLEFTBORDER, WITHRIGHTBORDER and WITHBOTTOMBORDER you can avoid this. Reason behind: sometimes you want a ROWAREA/COLAREA to be used as "neighbour" of other ROWAERA/COLAREA controls. In this case one of the "neighbours" has to avoid the rendering of border lines - otherwise two border lines will be rendered.	Optional	true false
withtopborder	See description of WITHLEFTBORDER property.	Optional	true false
withrightborder	See description of WITHLEFTBORDER property.	Optional	true false
withbottomborder	See description of WITHLEFTBORDER property.	Optional	true false
withtoppadding	The control by default renders some blank vertical space (2 pixels) on top of its area. Reason: if you vertically arrange one ROW/COLAREA after the other then automatically some distance is put between. By specifying "false" you can avoid this behaviour. "	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. specifying "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	
imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	

nameprop	Name of adapter property that provides as value the text that is shown inside the control.	Optional	
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.	Optional	true false

ROWAREAHEADER 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.	Optional	
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".	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	
Appearance			
align	Horizontal alignment of the controls inside the header line.	Optional	left center right

notabstop	The title of the area by default can be used by the user to hide/show the area's content. In order to also reach this title with the tab-key is part of the normal tab-sequence of a page. Set this property to "true" if you do not want to make the title reachable by tab-key. As consequence hiding/showing will only be available by mouse-clicking on the title.	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

ROWAREABODY Properties

Basic			
paddingleft	Number of pixels between the left border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
paddingright	Number of pixels between the right border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
bodystyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
withcontenttoppadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false
withcontentbottompadding	The control by default renders some blank vertical space (3 pixels) on bottom of the content area. By specifying "false" you can avoid this behaviour.	Optional	true false

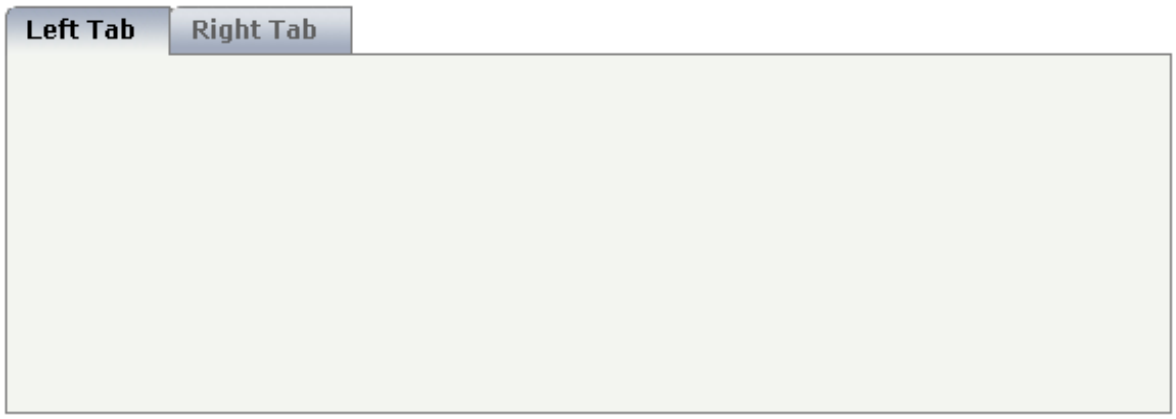
13 ROWTABAREA and COLTABAREA

The ROWTABAREA or COLTABAREA container is the representation of a tab control. A tab area consists of the ROWTABAREA or COLTABAREA definition. Inside this definition, you define TABPAGE containers representing the individual pages between which you can navigate.

Example:

```
<pagebody>
    <rowtabarea height="200" name1="Left Tab" page1="LEFT" name2="Right Tab" ↩
page2="RIGHT">
        <tabpage id="LEFT" takefullheight="true">
        </tabpage>
        <tabpage id="RIGHT" takefullheight="true">
        </tabpage>
    </rowtabarea>
</pagebody>
```

The above XML layout produces the following page:



Inside the ROWTABAREA definition, specify the name and the ID of each area you want to display. Pay attention to the naming of the `page*` properties: the name must not contain any blank spaces or non-alphanumeric characeters. Start the `page*` values with a character, not with a number.

Specify the individual toggle areas - by the TABPAGE definition. Each TABPAGE holds an ID which must be equal to the definition on ROWTABAREA level. Each TABPAGE has a `display` property which is set to "none" for all TABPAGE definitions except the first one.

Each TABPAGE is a container itself - i.e. inside the TABPAGE, place controls (or containers) by adding ITR or TR rows and place elements into these rows.

ROWTABAREA Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Obligatory	100 150 200 250 300 250 400 50% 100%
leftindent	Inserts a horizontal distance left of the first "tab" and shifts the "tabs" to the right as consequence. The value you may define represents the number of pixels that are inserted.	Optional	1 2 3 int-value
scrollable	If set to "true" then small icons will appear on the right border of the control. If the size of the "tabs" is too big and some tabs are cut as consequence then you can use these icons for scrolling left and right.	Optional	true false
name1	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Sometimes obligatory	
textid1	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Sometimes obligatory	

page1	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Obligatory	
withclose1	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name2	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid2	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page2	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose2	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name3	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid3	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page3	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below -	Optional	

	holding exactly the id that is defined in the PAGE property.		
withclose3	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name4	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid4	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page4	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose4	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name5	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid5	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page5	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose5	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false

name6	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid6	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page6	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose6	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name7	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid7	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page7	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose7	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name8	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid8	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	

page8	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose8	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name9	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid9	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page9	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose9	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name10	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid10	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page10	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below -	Optional	

	holding exactly the id that is defined in the PAGE property.		
withclose10	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name11	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid11	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page11	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose11	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name12	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid12	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page12	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose12	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false

name13	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid13	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page13	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose13	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name14	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid14	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page14	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose14	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name15	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid15	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	

page15	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose15	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
name16	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid16	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page16	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
withclose16	If you want a close-icon to be shown at the right top corner of the tab, set this property value to "true". Default is "false".	Optional	true false
Binding			
openedindexprop	Name of adapter property that represents the index of the "tab" that is currently opened. There are two ways of using the property: either you can define by the adapter property's value which "tab" should be opened. Or you can react inside your adapter object when the user does a "tab" selection (also have a look onto the property OPENMETHOD!). The property must be of type "int" or "Integer" (or "String"). The left most "tab" represents index "0", the next one "1", etc.	Optional	

openmethod	Name of the adapter method that is invoke when the user does a "tab" selection. The index of the "tab" that is opened can be transferred to the adapter by using the property OPENEDINDEXPROP.	Optional	
visibleprop1	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop2	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop3	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop4	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop5	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop6	Name of property that defines if the corresponding tag is visible or not. NOTICE: If	Optional	

	you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		
visibleprop7	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop8	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop9	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop10	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop11	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	

visibleprop12	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop13	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop14	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop15	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop16	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
disabledprop1	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	

disabledprop2	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop3	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop4	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop5	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop6	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop7	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop8	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop9	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In	Optional	

	COLTABAREA controls this property is only supported for IE.		
disabledprop10	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop11	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop12	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop13	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop14	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop15	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop16	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
titleprop1	Property of adapter that dynamically defines the title of the control. The title is displayed as tool	Optional	

	tip when the user moves the mouse onto the control.		
titleprop2	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop3	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop4	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop5	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop6	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop7	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop8	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop9	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop10	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop11	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop12	Property of adapter that dynamically defines the title of the control. The title is displayed as tool	Optional	

	tip when the user moves the mouse onto the control.		
titleprop13	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop14	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop15	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop16	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
tabselectedstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is not selected	Optional	background-color: #FF0000

	and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		color: #0000FF font-weight: bold
tabdisabledstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is disabled.	Optional	background-color: #FF0000

	NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		color: #0000FF font-weight: bold
tabselectedstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must	Optional	background-color: #FF0000 color: #0000FF

	also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		font-weight: bold
tabunselectedstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property	Optional	background-color: #FF0000 color: #0000FF

	OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		font-weight: bold
tabdisabledstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You	Optional	background-color: #FF0000 color: #0000FF

	don't have to set a value at runtime, but you need to specify a valid name.		font-weight: bold
tabselectedstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	don't have to set a value at runtime, but you need to specify a valid name.		
tabunselectedstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	value at runtime, but you need to specify a valid name.		
tabdisabledstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Appearance			
withleftborder	If specified as "false" then no left border will be drawn.	Optional	true false
withrightborder	If specified as "false" then no right border will be drawn.	Optional	true false
withbottomborder	If specified as "false" then no bottom border will be drawn.	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1
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

withtoppadding	The control by default renders some blank vertical space (2 pixels) on top of its area. Reason: if you vertically arrange one ROW/COLAREA after the other then automatically some distance is put between. By specifying "false" you can avoid this behaviour. "	Optional	true false
tabpagepaddingleft	Number of pixels between the left border and the area's content. Default is 5 pixels.	Sometimes obligatory	1 2 3 int-value
tabpagepaddingright	Number of pixels between the right border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
tabpagepaddingtop	Number of pixels between the top border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
tabpagepaddingbottom	Number of pixels between the bottom border and the area's content. Default is 5 pixels.	Optional	1 2 3 int-value
withflash	Adds animation effects when the user uses the control.	Optional	
Online Help			
title1	Tooltip text that appears on the corresponding tab.	Optional	
title2	Tooltip text that appears on the corresponding tab.	Optional	
title3	Tooltip text that appears on the corresponding tab.	Optional	

title4	Tooltip text that appears on the corresponding tab.	Optional	
title5	Tooltip text that appears on the corresponding tab.	Optional	
title6	Tooltip text that appears on the corresponding tab.	Optional	
title7	Tooltip text that appears on the corresponding tab.	Optional	
title8	Tooltip text that appears on the corresponding tab.	Optional	
title9	Tooltip text that appears on the corresponding tab.	Optional	
title10	Tooltip text that appears on the corresponding tab.	Optional	
title11	Tooltip text that appears on the corresponding tab.	Optional	
title12	Tooltip text that appears on the corresponding tab.	Optional	
title13	Tooltip text that appears on the corresponding tab.	Optional	
title14	Tooltip text that appears on the corresponding tab.	Optional	
title15	Tooltip text that appears on the corresponding tab.	Optional	
title16	Tooltip text that appears on the corresponding tab.	Optional	
titletextid1	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid2	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid3	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid4	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	

titletextid5	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid6	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid7	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid8	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid9	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid10	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid11	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid12	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid13	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid14	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid15	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	

titletextid16	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
Comment			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Miscellaneous			
testtoolid1	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	
testtoolid2	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	
testtoolid3	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	
testtoolid4	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	
testtoolid5	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	
testtoolid6	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	
testtoolid7	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	
testtoolid8	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	
testtoolid9	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	
testtoolid10	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	
testtoolid11	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	

testtoolid12	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	
testtoolid13	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	
testtoolid14	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	
testtoolid15	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	
testtoolid16	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	

COLTABAREA Properties

The properties of COLTABAREA are very similar to those of ROWTABAREA.

Basic			
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 percentage 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%
leftindent	Inserts a horizontal distance left of the first "tab" and shifts the "tabs" to the right as consequence.	Optional	1

	The value you may define represents the number of pixels that are inserted.		2 3 int-value
scrollable	If set to "true" then small icons will appear on the right border of the control. If the size of the "tabs" is too big and some tabs are cut as consequence then you can use these icons for scrolling left and right.	Optional	true false
name1	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Sometimes obligatory	
textid1	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Sometimes obligatory	
page1	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Obligatory	
name2	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid2	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page2	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name3	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	

textid3	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page3	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name4	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid4	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page4	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name5	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid5	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page5	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	

name6	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid6	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page6	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name7	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid7	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page7	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name8	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid8	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page8	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below -	Optional	

	holding exactly the id that is defined in the PAGE property.		
name9	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid9	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page9	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name10	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid10	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page10	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name11	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid11	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page11	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters.	Optional	

	For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.		
name12	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid12	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page12	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name13	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid13	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page13	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name14	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid14	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page14	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters.	Optional	

	For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.		
name15	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid15	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page15	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
name16	Text that is shown in the corresponding "tab". Either define the text as NAME or as language dependent TEXTID.	Optional	
textid16	Text ID that is transferred in a corresponding literal at runtime by the multi language management.	Optional	
page16	Id of the TABPAGE that is defined as child of the TABAREA. Use an id that is unique within the page and that is a "healthy" id: starting with characters, without blanks and without "strange" characters. For each "tab" of the TABAREA you have to create one corresponding TABPAGE below - holding exactly the id that is defined in the PAGE property.	Optional	
Binding			
openedindexprop	Name of adapter property that represents the index of the "tab" that is currently opened. There are two ways of using the property: either you can define by the adapter property's value which "tab" should be opened. Or you can react inside your adapter object when the user does a "tab" selection (also have a look onto the property OPENMETHOD!).	Optional	

	The property must be of type "int" or "Integer" (or "String"). The left most "tab" represents index "0", the next one "1", etc.		
openmethod	Name of the adapter method that is invoke when the user does a "tab" selection. The index of the "tab" that is opened can be transferred to the adapter by using the property OPENEDINDEXPROP.	Optional	
visibleprop1	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop2	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop3	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop4	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop5	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a	Optional	

	value at runtime, but you need to specify a valid name.		
visibleprop6	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop7	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop8	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop9	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop10	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop11	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute	Optional	

	OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.		
visibleprop12	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop13	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop14	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop15	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
visibleprop16	Name of property that defines if the corresponding tag is visible or not. NOTICE: If you want the framework to automatically set the focus to the first visible tab you also must apply a name for the attribute OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	
disabledprop1	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In	Optional	

	COLTABAREA controls this property is only supported for IE.		
disabledprop2	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop3	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop4	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop5	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop6	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop7	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop8	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop9	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at	Optional	

	runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.		
disabledprop10	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop11	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop12	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop13	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop14	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop15	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	
disabledprop16	Name of the adapter parameter that dynamically defines if the control is disabled or enabled at runtime. If the value at runtime is set to TRUE the control is visible but disabled. In COLTABAREA controls this property is only supported for IE.	Optional	

titleprop1	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop2	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop3	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop4	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop5	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop6	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop7	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop8	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop9	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop10	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop11	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	

titleprop12	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop13	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop14	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop15	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
titleprop16	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
tabselectedstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop1	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabunselectedstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop2	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop3	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabdisabledstyleprop4	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop5	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop6	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabselectedstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop7	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop8	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabunselectedstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop9	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop10	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabdisabledstyleprop11	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop12	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop13	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabselectedstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop14	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabunselectedstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop15	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabselectedstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is selected. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

tabunselectedstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is not selected and not disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
tabdisabledstyleprop16	Name of the adapter parameter that dynamically defines the style for a tab which is disabled. NOTICE: When using this property you must also set the property OPENEDINDEXPROP. You don't have to set a value at runtime, but you need to specify a valid name.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Appearance			
withleftborder	If specified as "false" then no left border will be drawn.	Optional	
withrightborder	If specified as "false" then no right border will be drawn.	Optional	
withbottomborder	If specified as "false" then no bottom border will be drawn.	Optional	
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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1
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
Online Help			

title1	Tooltip text that appears on the corresponding tab.	Optional	
title2	Tooltip text that appears on the corresponding tab.	Optional	
title3	Tooltip text that appears on the corresponding tab.	Optional	
title4	Tooltip text that appears on the corresponding tab.	Optional	
title5	Tooltip text that appears on the corresponding tab.	Optional	
title6	Tooltip text that appears on the corresponding tab.	Optional	
title7	Tooltip text that appears on the corresponding tab.	Optional	
title8	Tooltip text that appears on the corresponding tab.	Optional	
title9	Tooltip text that appears on the corresponding tab.	Optional	
title10	Tooltip text that appears on the corresponding tab.	Optional	
title11	Tooltip text that appears on the corresponding tab.	Optional	
title12	Tooltip text that appears on the corresponding tab.	Optional	
title13	Tooltip text that appears on the corresponding tab.	Optional	
title14	Tooltip text that appears on the corresponding tab.	Optional	
title15	Tooltip text that appears on the corresponding tab.	Optional	
title16	Tooltip text that appears on the corresponding tab.	Optional	
titletextid1	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid2	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid3	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management	Optional	

	replaces the textid with a language dependent literal.		
titletextid4	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid5	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid6	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid7	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid8	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid9	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid10	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid11	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid12	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid13	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid14	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management	Optional	

	replaces the textid with a language dependent literal.		
titletextid15	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
titletextid16	Text ID for the tooltip of the corresponding "tab". At runtime the multi language management replaces the textid with a language dependent literal.	Optional	
Comment			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

TABPAGE Properties

Basic			
id	Id of the TABPAGE. Each page has an id that refers to the PAGE1 .. PAGE9 definition inside the ROW/COLTABAREA control that contains the TABPAGE. Clicking a "tab" will display the TABPAGE with the associated id.	Obligatory	
display	Initial display status of the TABPAGE. The first TABPAGE inside the ROW/COLTABAREA control must be set to "". All others need to be set to "none". - If a ROW/COLTABAREA should show up with two or more pages being visible one below the other then check the setting of this property!"	Sometimes obligatory	
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	Optional	true false
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control.	Optional	true false

	If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

The Most Common Error

Do you receive JavaScript errors when clicking in the tabs? Then take a further look at the ID assignments in the ROWTABAREA or COLTABAREA control on the one hand, and in the TABPAGE control on the other hand: each `page*` property of a ROWTABAREA or COLTABAREA defines an ID that must exactly match an `id` property of TABPAGE.

If you have more than one ROWTABAREA or COLTABAREA inside your page: do not use the same IDs - each ID must be unique throughout one page.

Example: Controlling which Tab is displayed by the Server Adapter

The following example demonstrates the usage of the property `openedindexprop` on ROWTABAREA level:

The user selects the value of the property `index` using the combo control. The `index` property controls also which tab is displayed inside the ROWTABAREA control.

The layout definition is as follows:

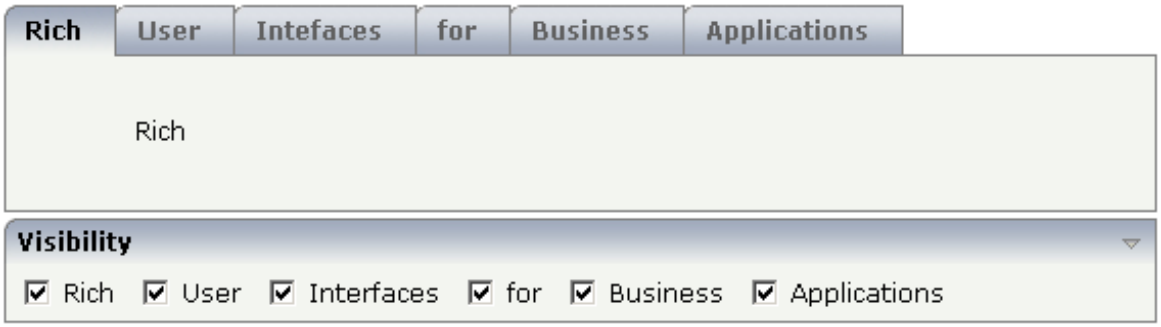
```
<pagebody>
    <rowarea name="Dynamic setting of index in TABAREA">
        <itr>
            <label name="Index" width="100">
            </label>
            <combofix valueprop="index" size="1" flush="server">
                <combooption name="First  (=0)" value="0">
                </combooption>
                <combooption name="Second  (=1)" value="1">
                </combooption>
                <combooption name="Third (=2)" value="2">
                </combooption>
            </combofix>
        </itr>
    </rowarea>
    <rowtabarea height="200" openedindexprop="index"
                name1="First" page1="FIRST"
                name2="Second" page2="SECOND"
                name3="Third" page3="THIRD">
        <tabpage id="FIRST">
        </tabpage>
        <tabpage id="SECOND">
        </tabpage>
        <tabpage id="THIRD">
        </tabpage>
    </rowtabarea>
</pagebody>
```

The adapter class on the server side looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class TabAreaIndexAdapter extends Adapter
{
    // property >index<
    int m_index;
    public int getIndex() { return m_index; }
    public void setIndex(int value) { m_index = value; }
}
```

Example: Controlling the Visibility of Tab Pages

For each individual tab page, you can control at runtime whether it is visible or not. The following example allows the user to control the visibility of tabs using check boxes:



The XML layout is:

```
<rowtabarea height="100" name1="Rich" page1="RICH" visibleprop1="page1Visibility"
                         name2="User" page2="USER" visibleprop2="page2Visibility"
                         name3="Intefaces" page3="INTERFACES" ↩
visibleprop3="page3Visibility"
                         name4="for" page4="FOR" visibleprop4="page4Visibility"
                         name5="Business" page5="BUSINESS" ↩
visibleprop5="page5Visibility"
                         name6="Applications" page6="APPLICATIONS"
                                              visibleprop6="page6Visibility">
    <tabpage id="RICH">
        <vdist height="20">
        </vdist>
        <itr>
            <hdist width="60">
            </hdist>
            <label name="Rich" asplaintext="true" textalign="center">
            </label>
        </itr>
```

```
    </tabpage>
    <tabpage id="USER">
        ...
    </tabpage>
    ...
    ...
    ...
<rowarea name="Visibility">
    <itr>
        <checkbox valueprop="page1Visibility" flush="server">
        </checkbox>
        <hdist>
        </hdist>
        <label name="Rich" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
        <checkbox valueprop="page2Visibility" flush="server">
        </checkbox>
        <hdist>
        </hdist>
        <label name="User" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
        <checkbox valueprop="page3Visibility" flush="server">
        </checkbox>
        <hdist>
        </hdist>
        <label name="Interfaces" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
        <checkbox valueprop="page4Visibility" flush="server">
        </checkbox>
        <hdist>
        </hdist>
        <label name="for" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
        <checkbox valueprop="page5Visibility" flush="server">
        </checkbox>
        <hdist>
        </hdist>
        <label name="Business" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
        <checkbox valueprop="page6Visibility" flush="server">
        </checkbox>
        <hdist>
```

```
        </hdist>
        <label name="Applications" asplaintext="true">
        </label>
        <hdist width="10">
        </hdist>
    </itr>
</rowarea>
```

You see that the definition of the properties that control the visibility of tab pages is done in the ROWTABAREA (not on TABPAGE level). The check boxes reference the same adapter properties as used on ROWTABAREA level.

In the adapter code, the corresponding boolean properties are provided:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class VisibilityOfTabPagesAdapter
    extends Adapter
{
    // property >page1Visibility<
    boolean m_page1Visibility;
    public boolean getPage1Visibility() { return m_page1Visibility; }
    public void setPage1Visibility(boolean value) { m_page1Visibility = value; }

    // property >page2Visibility<
    boolean m_page2Visibility;
    public boolean getPage2Visibility() { return m_page2Visibility; }
    public void setPage2Visibility(boolean value) { m_page2Visibility = value; }

    // property >page3Visibility<
    boolean m_page3Visibility;
    public boolean getPage3Visibility() { return m_page3Visibility; }
    public void setPage3Visibility(boolean value) { m_page3Visibility = value; }

    // property >page4Visibility<
    boolean m_page4Visibility;
    public boolean getPage4Visibility() { return m_page4Visibility; }
    public void setPage4Visibility(boolean value) { m_page4Visibility = value; }

    // property >page5Visibility<
    boolean m_page5Visibility;
    public boolean getPage5Visibility() { return m_page5Visibility; }
    public void setPage5Visibility(boolean value) { m_page5Visibility = value; }

    // property >page6Visibility<
    boolean m_page6Visibility;
    public boolean getPage6Visibility() { return m_page6Visibility; }
    public void setPage6Visibility(boolean value) { m_page6Visibility = value; }
}
```

Note: In the previous example, the `openedindexprop` property of the ROWTABAREA was used. Be aware of the fact that each tab page still keeps its stable index position - no matter whether it is displayed or not.

14 ROWTABLE0 and COLTABLE0

The ROWTABLE0 or COLTABLE0 container is not visible. Normally, it is just used for arranging controls. The following example shows how to define two columns - inside a ROWAREA - to arrange controls:

```
<pagebody>
    <rowarea name="Area 1">
        <itr takefullwidth="true">
            <coltable0 width="50%" takefullheight="true">
                <itr>
                    <label name="Factor 1" width="100">
                    </label>
                    <field valueprop="factor1" length="5">
                    </field>
                </itr>
            </coltable0>
            <coltable0 width="50%" takefullheight="true">
                <itr>
                    <label name="Factor 2" width="100">
                    </label>
                    <field valueprop="factor2" length="5">
                    </field>
                </itr>
            </coltable0>
        </itr>
    </rowarea>
</pagebody>
```

The result looks as follows:

Inside the ROWAREA, two COLTABLE0 tags are placed - each occupying 50% of the width. Each COLTABLE0 area builds - independently from the other - its own table rows (ITR rows in the example).

All complex field arrangements should be done by using ROWTABLE0/COLTABLE0 tags as shown in the example.

ROWTABLE0 Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
align	Alignment of the content of the ITR row. Background: the ITR as independent table row renders a table into its content area. Inside this table a row is opened in which the controls are placed. This table normally is starting on the left of the ITR row. With this ALIGN property you can explicitly define the alignment of the table.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut.	Optional	true false

	You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
flashprop	Name of server side property/data element that triggers a "flashing" of the area. "Flashing" means that the area is animated for a short point of time in order to make the user that e.g. some change of data happened inside the area. The server side property/data element is an index - whenever you change the index then a flashing of the control is triggered on client side. Pay attention: do not mix the "flashing" of an area with the "flushing" of controls - "flushing" is the way an input control (e.g. field) triggers server side updates when the user changed the value, "flashing" is pure animation.	Optional	

COLTABLE0 Properties

The properties for COLTABLE0 are very similar to those of ROWTABLE0.

Basic			
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 percentage 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%
widthprop	Name of adapter property that dynamically defined the height of the control. Must return a valid width.	Optional	
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	Optional	true false
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as	Optional	true false

	normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

15 ROWDYNAVIS and COLDYNAVIS

The ROWDYNAVIS or COLDYNAVIS container is used to add dynamic reaction to your layout.

The container is not visible - similar to the TABLE0 container. What is the difference? You control the appearance of the container by an adapter property. Have a look at the following example.

If you enter "United States" as a country, the input line for the state will appear under the input line for the country:

The XML code looks as follows:

```
<rowarea name="Address Input">
    <itr>
        <label name="Country" width="100">
        </label>
        <field valueprop="country" flush="true" length="30">
        </field>
    </itr>
    <rowdynavis valueprop="visible">
        <itr>
            <label name="State" width="100">
            </label>
            <field valueprop="state" length="30">
            </field>
        </itr>
    </rowdynavis>
</rowarea>
```

A ROWDYNAVIS container is placed inside the ROWAREA container. The ROWDYNAVIS container is bound to the adapter class property `visible`.

The adapter class looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class RowTable0Adapter
    extends Adapter
{
    // property >factor1<
    String m_factor1;
    public String getFactor1() { return m_factor1; }
    public void setFactor1(String value) { m_factor1 = value; }

    // property >factor2<
    String m_factor2;
    public String getFactor2() { return m_factor2; }
    public void setFactor2(String value) { m_factor2 = value; }

    // property >country<
    String m_country;
    public String getCountry() { return m_country; }
    public void setCountry(String value) { m_country = value; }

    // property >state<
    String m_state;
    public String getState() { return m_state; }
    public void setState(String value) { m_state = value; }

    // property >visible<
    boolean m_visible = true;
    public boolean getVisible()
    {
        if (m_country != null && m_country.equalsIgnoreCase("United States"))
            return true;
        else
            return false;
    }
    public void setVisible(boolean value) { m_visible = value;}
}
```

The property `visible` depends on the user input of the `country` property. It returns a boolean value. Since a ROWDYNAVIS area is a container, it can contain rows - with input lines - and containers. Therefore, you can add dynamic reaction to your layout definitions in a very flexible way.

ROWDYNAVIS Properties

Basic			
valueprop	Name of adapter property that defines if the area is visible ("true") or invisible ("false"). Must be of type "boolean" / "Boolean".	Obligatory	
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
style	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires	Optional	true false

	to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		

COLDYNAVIS Properties

The properties of COLDYNAVIS are very similar to those of ROWDYNAVIS.

Basic			
valueprop	Name of adapter property that defines if the area is visible ("true") or invisible ("false"). Must be of type "boolean" / "Boolean".	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 percentage 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%

takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	Optional	true false
style	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster	Optional	true false

	in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Some Comments on Controlling the Visibility of Controls

ROWDYNAVIS and COLDYNAVIS are container controls that are explicitly defined to provide an area which can be explicitly switched on and off. In addition you will later on see that many controls can control their visiblity and their input status by themselves. For example, a FIELD control can specify if it is invisible, editable, holding an error input etc. in a dynamic way. You may also have noticed that an ITR row definition has an associated `visibleprop` property - linking to a data property that dynamically controls the visibility of the row at runtime.

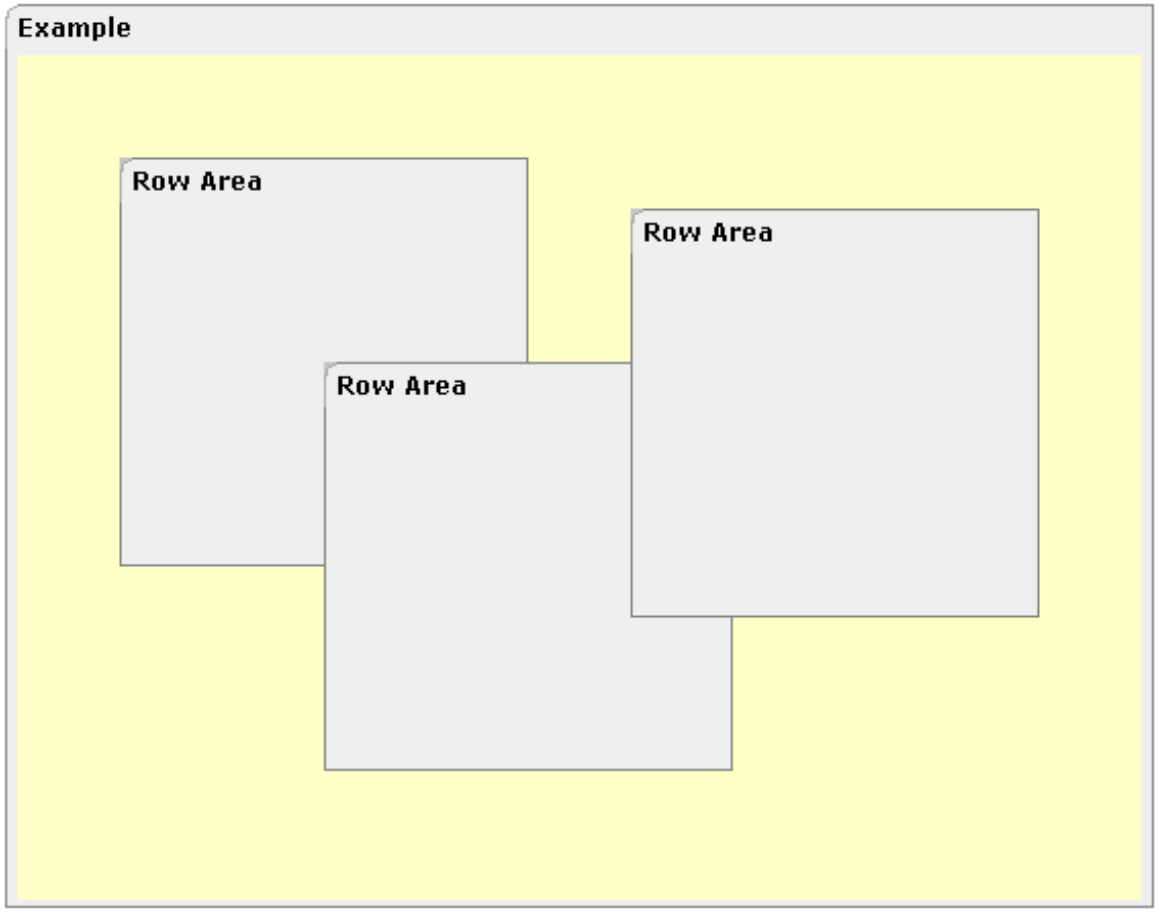
Use ROWDYNAVIS and COLDYNAVIS for explicitly defining container areas to be switched on/off. Use the control's binding to properties to do the fine-granular control of visibility inside one container.

A bad example of usage would be if you place a COLDYNAVIS container around each FIELD that you want to control in means of visibility. Use the FIELD's `statusprop` property instead.

16 ROWDIV and INNERDIV

The ROWDIV container represents an area with a defined size. Inside this area you can arrange INNERDIV containers. The INNERDIV containers have a defined x-, y- and z-position inside the ROWDIV area, and they have a defined width and height. INNDERDIV containers can overlap; by using the z-position, you can define which INNERDIV container is on top of which other INNERDIV container. Inside an INNERDIV container, you can arrange any other container or control - just as with normal containers.

Have a look at the following example:

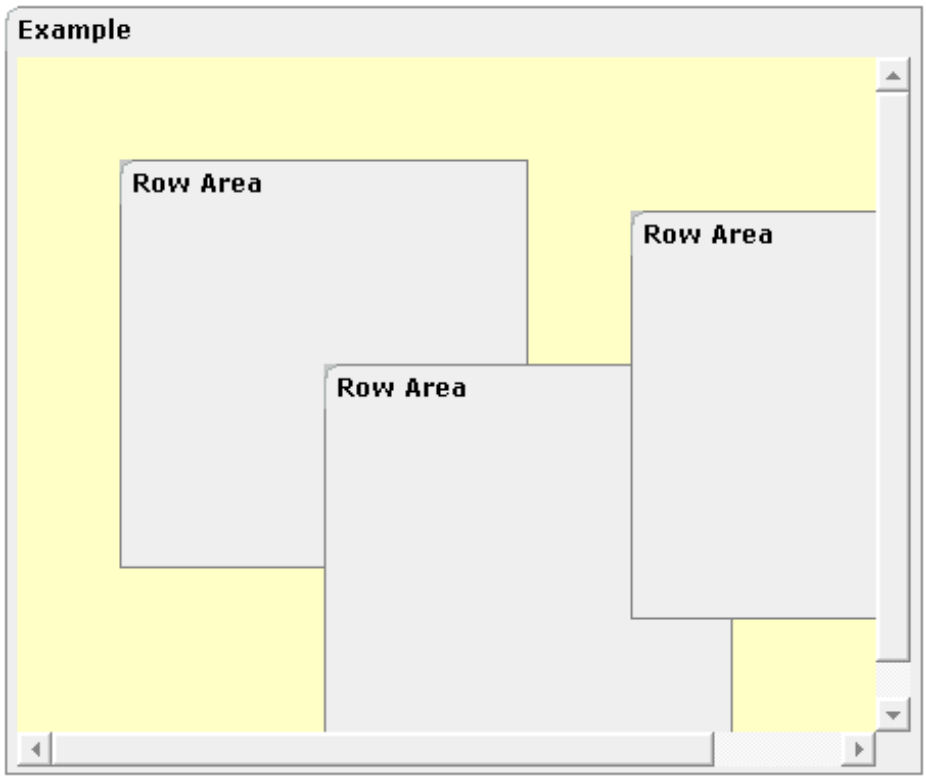


Inside a ROWAREA container, a ROWDIV container is arranged. Inside the ROWDIV container, three INNERDIV containers are arranged - each one holding a ROWAREA.

The XML layout definition looks as follows:

```
<rowarea name="Example" height="100%">
    <rowdiv height="100%" style="background-color: #FFFFC0">
        <innerdiv width="200" height="200" zindex="99" left="150" top="150"
                  style="background-color: #C0C0C0">
            <rowarea name="Row Area" height="100%" withtoppadding="false">
            </rowarea>
        </innerdiv>
        <innerdiv width="200" height="200" zindex="98" left="50" top="50"
                  style="background-color: #C0C0C0">
            <rowarea name="Row Area" height="100%" withleftborder="true" ↩
withtopborder="true"
                     withrightborder="true" withbottomborder="true" ↩
withtoppadding="false">
            </rowarea>
        </innerdiv>
        <innerdiv width="200" height="200" zindex="100" left="300" top="75"
                  style="background-color: #C0C0C0">
            <rowarea name="Row Area" height="100%" withtoppadding="false">
            </rowarea>
        </innerdiv>
    </rowdiv>
</rowarea>
```

If the ROWDIV area is too small to hold the INNERDIV containers, then the ROWDIV area starts scrolling:



When to Use ROWDIV and INNERDIV Containers

The typical usage scenarios of ROWDIV and INNERDIV containers is:

- when you want to place a certain area at a certain position on the screen - without wanting to explicitly define VDIST/HDIST elements;
- when you want to explicitly work with overlapping areas.

Note that the parallel usage of pixel and percentage sizing is not supported with ROWDIV and INNERDIV in the same way as supported with normal containers (for example, ROWAREA and COLAREA). With normal containers, you can specify scenarios like the following: the left container occupies 200 pixels, the right container occupies 100%. The table rendering is clever enough to render the result accordingly. With INNERDIV containers, the percentage definitions are always in relation to the height and width of the surrounding ROWDIV control.

Consequence: Do not use ROWDIV and INNERDIV for the basic structuring of containers inside your page, but only use them for the two usage aspects mentioned before.

ROWDIV Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Obligatory	100 150 200 250 300 250 400 50% 100%
style	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000	Optional	

	background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
divclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

INNERDIV Properties

Basic			
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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20").	Obligatory	100 150 200 250 300

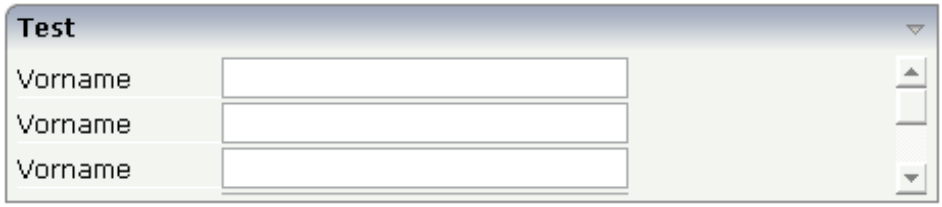
	(C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		250 400 50% 100%
left	Left position of control. Either define a pixel value ("100") or a percentage value ("30%").	Obligatory	
top	Top position of control. Either define a pixel value ("100") or a percentage value ("30%").	Obligatory	
zindex	Z-index of the control. If two controls overlap then the one with the higher z-index is drawn in front of the other one.	Optional	1 2 3 int-value
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
leftdistance	If set to "true" then a small distance (3px) is kept between the left border of the control and its content. Default is "false".	Optional	true false
rightdistance	If set to "true" then a small distance (3px) is kept between the right border of the control and its content. Default is "false".	Optional	true false
style	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
Binding			
widthprop	Name of adapter property that dynamically defined the width of the control. Must return a valid width.	Optional	

leftprop	Name of adapter property that dynamically defines the left position of the control. Must return a valid value for 'left position'.	Optional	
dropwidthprop	Name of the adapter property that dynamically defines the width of the drop target.	Optional	
dropoffsetprop	Name of the adapter property that dynamically defines the offset used for the drop target.	Optional	
dropmethod	Method which is called in the Adapter when the drop operation is finished.	Optional	

17 ROWSCROLLAREA

The ROWSCROLLAREA represents a container area with a certain size. The container is not visible. If the contents of the container area exceed the size of the container area, then scroll bars are added accordingly.

Have a look at the following example:



Inside a normal ROWAREA with the title "Test", a ROWSCROLLAREA is positioned. Inside the ROWSCROLLAREA, a number of lines is arranged so that the total height of the lines exceeds the height of the ROWSCROLLAREA. Consequently, a vertical scroll bar is shown on the right.

The XML layout looks as follows:

```
<rowarea name="Test" height="100">
    <rowscrollarea height="100%">
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
        </itr>
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
        </itr>
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
        </itr>
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
        </itr>
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
```

```
        </itr>
        <itr>
            <label name="Vorname" width="100">
            </label>
            <field valueprop="firstname" width="200">
            </field>
        </itr>
    </rowscrollarea>
</rowarea>
```

ROWSCROLLAREA Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Obligatory	100 150 200 250 300 250 400 50% 100%
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true"	Optional	true false

	the table itself is sized to fill the maximum height of the available area.		
takefullwidth	If set to "true" then the control takes all available horizontal width as its width. If set to "false" then the control does not have a predefined width but grows with its content.	Optional	true false
areastyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
areaclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much	Optional	true false

	faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.		
vscrollposprop	Name of adapter property that dynamically sets the position of the vertical scrollbar. The value top will scroll up to the top, the value bottom will scroll down to the bottom of the container.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "hidden".	Optional	auto scroll hidden

18 HSPLIT and VSPLIT

HSPLIT or VSPLIT allows to define a container area that is subdivided into two split cells. Between the split cells there is a border. By dragging and dropping the border, you can change the size of the split cells. Each split cell itself is a container that can be used just as normal.

While an HSPLIT control subdivides an area into two split cells by a horizontal line, VSPLIT uses a vertical line.

Example for HSPLIT

The following example shows the usage of the HSPLIT control:

The split area is divided into two cells: a green cell and a red cell. In addition, there is a line at the bottom in which you can provide the split factor.

The XML layout definition is:

```
<rowarea name="HSPLIT Control" height="100%">
    <hsplit height="100%" heighttopprop="heighttop" hsplitstyle="border:1 solid ↩
#000000">
        <splitcell takefullheight="true" cellstyle="background-color: #00FF00">
            <tr height="100%">
                <label name="Top Split Cell" asplaintext="true">
                </label>
            </tr>
        </splitcell>
        <splitcell takefullheight="true" cellstyle="background-color: #FF0000">
            <tr height="100%">
                <label name="Bottom Split Cell" asplaintext="true">
```

```
                </label>
            </tr>
        </splitcell>
    </hsplit>
    <vdist>
    </vdist>
    <itr>
        <hdist width="100%">
        </hdist>
        <label name="Set Top Height" width="100">
        </label>
       <field valueprop="heighttop" width="100" flush="server" validation="[0-9%]+"
              validationuserhint="100, 200, 500, 30%, 50%">
        </field>
    </itr>
</rowarea>
```

You see that the vertical split area consists of

- one VSPLIT definition, and
- two SPLITCELL definitions.

It is not allowed to have more than two split cells inside one HSPLIT container.

The sizing of the split cells can be done by using a property that is referenced by the HSPLIT property `heighttopprop`. The property must return either a percentage value or a pixel value. When the user changes the size by moving the line between the split cells, then the current new pixel width of the left split cell is written back into the property.

Example for VSPLIT

The VSPLIT control is defined in the same way as the HSPLIT control - but now transferred to vertical dimension. It looks like:

The VSPLIT part of the XML layout definition is:

```
<vsplit height="200" widthleftprop="widthleft" vsplitstyle="border: 1 solid #000000">
    <splitcell takefullheight="true" cellstyle="background-color:#00FF00">
        <itr>
            <label name="Left Split Cell" asplaintext="true">
            </label>
        </itr>
    </splitcell>
    <splitcell takefullheight="true" cellstyle="background-color: #FF0000">
        <itr>
            <label name="Right Split Cell" asplaintext="true">
            </label>
        </itr>
    </splitcell>
</vsplit>
```

HSPLIT Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20").	Optional	100 150 200 250 300 250

	(C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		400 50% 100%
heighttop	Definition of the initial height of the top split area. The height either is a pixel value ("100") or a percentage value ("50%"). You can also define the height dynamically by your adapter - see documentation for HEIGHTTOPPROP property.	Optional	1 2 3 int-value
heighttopprop	Name of adapter property that specifies the height of the top split area. The adapter property must return either a pixel value ("100") or a percentage value ("50%").	Optional	
hsplitstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Optional	auto scroll hidden

VSPLIT Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
widthleftprop	Name of adapter property that specifies the width of the left split area. The adapter property must return either a pixel value ("100") or a percentage value ("50%").	Optional	
vsplitstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
overflow	Definition of the vertical scrollbar's appearance.	Optional	auto

	You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".		scroll hidden

SPLITCELL Properties

Basic			
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.	Optional	true false
cellstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Defining the Split Size

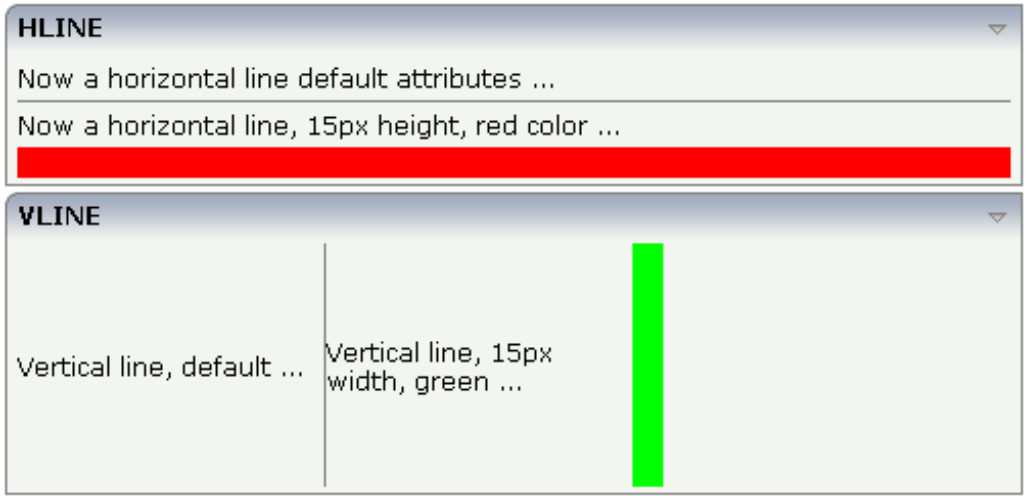
The split size of HSPLIT and VSPLIT can be set in the following ways:

- Fixed definition if initial split size: by using the HSPLIT property `heighttop` and the VSPLIT property `widthleft`, you can preset the size in a “hard way”. The value will be used as the initial size.
- By using the HSPLIT property `heighttopprop` and the VSPLIT property `widthleftprop`, the size can be defined by a server side property. Maybe you have some personalization in which the size is kept for every split area - and proposed the next time the user visits the page.

19 HLINE and VLINE

Both controls are actually not container controls, but they are typically used for structuring content - this is the reason why they are mentioned here. The controls are rather simple: they represent lines. HLINE represents a horizontal line and VLINE represents a vertical line.

Have a look at this demo:



The corresponding XML layout definition is:

```
<rowarea name="HLINE">
    <itr>
       <label name="Now a horizontal line default attributes ..." asplaintext="true">
        </label>
    </itr>
    <hline>
    </hline>
    <itr>
        <label name="Now a horizontal line, 15px height, red color ..." ↩
asplaintext="true">
        </label>
    </itr>
    <hline height="15" color="#FF0000">
    </hline>
</rowarea>
<rowarea name="VLINE" height="150">
    <itr height="100%">
        <label name="Vertical line, default ..." width="150" asplaintext="true">
        </label>
        <vline>
        </vline>
        <label name="Vertical line, 15px width, green ..." width="150" ↩
asplaintext="true">
        </label>
        <vline width="15" color="#00FF00">
        </vline>
```

```
    </itr>
</rowarea>
```

For each line, you can define its width/height and its color.

VLINE Properties

Basic			
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 percentage 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	
color	Color of the control. Value must follow format "#rrggbb", e.g. #000000 for black.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

HLINE Properties

Basic			
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	
color	Color of the control. Value must follow format "#rrggbb", e.g. #000000 for black.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

20 Performance Optimization with Containers

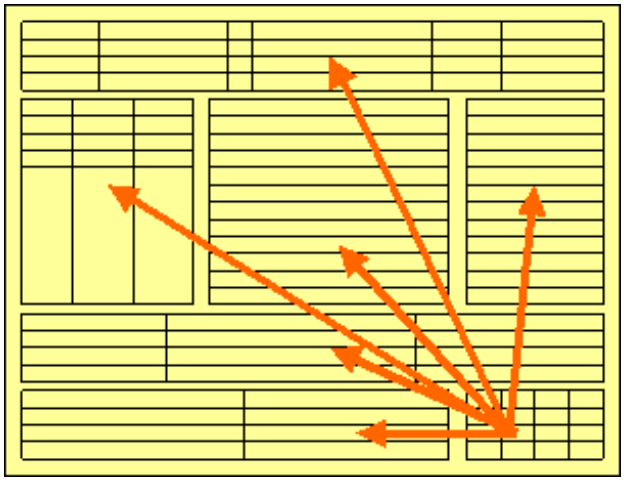
Containers internally use HTML table rendering for arranging their content: inside a container there are rows, inside the rows there are columns and inside the columns there are controls.

HTML table rendering is very powerful: if you have already written pages on your own using an HTML editor, then you know that you can size the container in the following way:

```
<table width='100%'>
<tr>
  <td width='100'>Hallo</td>
  <td width='100%'>Hello world!</td>
  <td><img src='xyz.gif'></td>
</tr>
</table>
```

During rendering time, the browser tries to optimize the table rendering. The browser knows that inside the definitions there is one column that wants to occupy the whole width, one column that wants to have a width of 100 pixels and one column that holds an image. Consequently, it somehow renders the table so that the best result is rendered. This optimization is quite expensive - especially if you have tables nested in tables nested in tables etc.

In nested table scenarios, every little change in one table can have the consequence that the whole HTML table is optimized again.



Since the optimization now happens on several levels, the browser uses a lot of resources to do so. This can be noticed especially if you render pages with a height of 100%: the page is not built by appending one information after the other - but you tell that the controls occupy a certain percentage based height of the whole page.

How can you find that out? If you have got the feeling that a page behaves in a slow way and you are not sure whether it is your server side application or the browser side rendering, then there are two ways to easily find out:

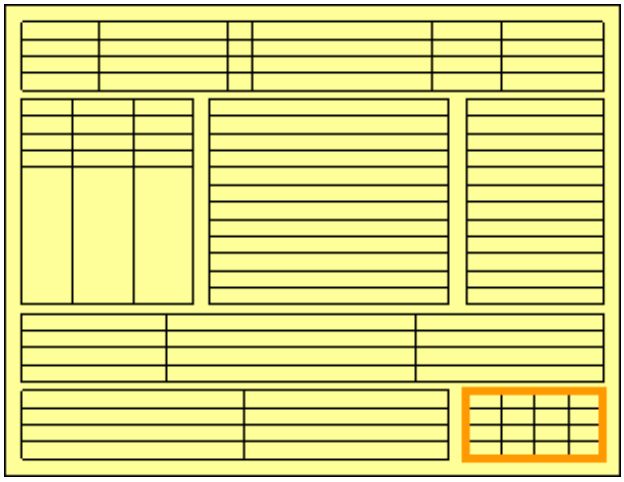
- Look into the Application Designer log file. Each server side request is recorded with its consumption of milliseconds on server side.
- Resize the page in the browser: if this is not fast but takes time, then this is an indicator for bad rendering performance - or in other words: for a lot of optimization that is happening behind the curtain.

But: there are nice ways to speed up the rendering - and to build optimization limits for the browser. Internally, the ways are quite simple, but the consequence can be dramatic.

Most containers support a `fixlayout` property: the possible values are "true" or "false" - "false" being the default. When switching the `fixlayout` property to "true", then the content area of the container is internally arranged in such a way that the area always determines its size from its own width and height specification. The browser does not look into the contents of the area in order to try to optimize the size of the area, but always follows the width and height that you define.

What happens if the controls inside your container area do not fit into the area? What does not fit inside the container area, is cut.

Setting `fixlayout` to "true" means that the browser only optimizes table rendering inside the container - but never outside - because the container has a certain size:



Follow the rules:

- Every time the size of a container area is not determined by its content but is explicitly set by you, switch the `fixlayout` flag to "true".
- The flag only has consequences if you define the width and height of the corresponding container. In cases in which the width is defined by the control (for example, ROWAREA always has a width of 100%), you have to define the height. The height is either defined by a corresponding `height` property or by a `takefullheight` property.

III Working with Controls

Controls are the elements that are placed inside containers. This part first gives some common rules that are valid for all controls, then describes the controls in more detail.

The information provided in this part is organized under the following headings:

STRIPSEL
SUBPAGE
TABSEL
TABSTRIP2
TAGCLOUD
TEXT
TEXTOUT
TOGGLE

Special Controls:

ACTIVEX (Deprecated)
GOOGLEMAP2
HELPICON
REPORT
ROWCHARTAREA
TIMER

21 Some Common Rules for all Controls

Name and Text ID

Every time a control needs a static text definition (the name of a button or the name of a label), there are always two possibilities to define this text:

- Specify a name directly.
- Specify a text ID. This is a literal replaced with a string that is determined inside the multi language management at runtime. For more details, see *Multi Language Management*.

Table, Row, Column, Control

Most controls that allow dynamic sizing offer the following properties:

- `colspan` - number of columns occupied by the control.
- `rowspan` - number of rows occupied by the control.
- `width` - width.
- `height` - height.

These properties influence the way how controls are placed into container rows.

For users of previous releases: the `width` property is deprecated. The rendering can be controlled by COLSPAN and ROWSPAN now. In addition, there is the CELLSPAN control allowing you to group your controls.

Explicit Alignment

Controls are put into table columns. If the column is wider or higher than the control itself, then you can explicitly control the vertical and horizontal alignment of the control inside the columns.

Most controls offer two properties:

- `valign`
  Specifies the vertical alignment. Valid values are "top", "middle", "bottom". "middle" is the default value.
- `align`
  Specifies the horizontal alignment. Valid values are "left", "center", "right". The default value depends on the control. For example, labels are aligned "left" by default, the default for radio buttons is "center".

Pay attention: `valign` and `align` only affect the position of the control inside the column in which it is positioned if the column is larger than the control. If the column is exactly as wide and high as the control itself, which is the typical case, then they do not have any visual effects - and also need not be defined.

`align`/`valign` do not affect the control's internal alignment.

Binding to Adapter Properties

Most controls provide properties to specify the binding to the adapter processing. There is a naming convention, which is:

- The names of the properties which specify the binding to a property end with "prop".
- The names of the properties which specify the binding to a method end with "method".

The type of the property which is referenced by a control depends on the control itself:

- Most controls directly bind to simple properties - i.e. properties returning a simple data type: String, int, boolean, float, etc.
- More complex controls bind to a complex data type - e.g. a text grid binds to a TEXTGRIDCollection.

The type of the property is described with each control. See also *Appendix C - Data Types to be Used by Adapter Properties*.

Directly Influencing the Control Style

All controls that incorporate textual information - such as labels, buttons or fields - offer the possibility to influence directly the style that is used for displaying the information.

The normal style is derived from the definition inside a cascading style definition file (file *layout.css* inside the *html/general* directory of the server). Overwrite or enhance this style information for your controls by passing the style information inside the corresponding style properties.

The properties specifying the style information end with the suffix "style", e.g. there is a property `labelstyle` for the label tag. The value of the property can be any kind of a valid HTML style specification. If you want to change the display style of a label to be large and blue, define the label in the following way:

```
<label name="Test" width="150" labelstyle="font-size: 24pt; color: #0000FF">
</label>
```

See *Adapting the Look & Feel* in *Java Pages Development* for information on how to change the style of controls.

Dynamically Controlling the Visibility and the Display Status of Controls

It is possible to influence the visibility of all input controls (FIELD, BUTTON, etc.) by properties of the adapter processing.

For some of these controls there is a property `visibleprop`, specifying an adapter property that returns "true" or "false". By this, you can control whether you want to display the control within the client or not.

Input controls support a property `statusprop` and a property `displayprop`. Using the corresponding adapter properties, you can dynamically control the display status of the input control. The adapter property for the `statusprop` can return the following values:

INVISIBLE
ERROR
ERROR_NO_FOCUS
FOCUS
FOCUS_NO_SELECT

Note: INVISIBLE is only supported in controls that don't have a `visibleprop` property. Otherwise, use the `visibleprop` instead, provided it is supported.

Responsive controls additionally support the following values:

WARNING
SUCCESS

Note: For SUCCESS a green and for warning an orange border color is used. You can customize the rendering via the css classes `has-warning` and `has-error`. For more information see the *Style Guide* in the *NaturalAjaxDemos*.

The adapter property for the `displayprop` specifies whether the control is display-only (TRUE) or whether it can be edited (FALSE). The adapter property can return the values "TRUE" and "FALSE".

The combination of these two property values dynamically defines how the controls are rendered at runtime (for an example, see **80_displayprop** in the **cisdemos** project). The following table defines the rendering of the control for the different combinations:

displayprop	statusprop	Control Status
FALSE (default)		EDIT
FALSE (default)	EDIT (deprecated) [1]	EDIT
FALSE (default)	INVISIBLE	INVISIBLE
FALSE (default)	ERROR	ERROR
FALSE (default)	ERROR_NO_FOCUS	ERROR_NO_FOCUS
FALSE (default)	FOCUS	FOCUS
FALSE (default)	FOCUS_NO_SELECT	FOCUS_NO_SELECT
TRUE		DISPLAY
TRUE	DISPLAY (deprecated) [1]	DISPLAY
TRUE	INVISIBLE	INVISIBLE
TRUE	ERROR	ERROR_DISPLAY
TRUE	ERROR_NO_FOCUS	ERROR_DISPLAY
TRUE	FOCUS	DISPLAY
TRUE	FOCUS_NO_SELECT	DISPLAY

[1] For `statusprop`, the above-mentioned deprecated values are still supported to ensure compatibility with older versions. In case you use these deprecated values for `statusprop`, the values for `displayprop` are ignored.

The literals used for the `statusprop` are available in the interface `com.softwareag.cis.util.IControlStatusConstants`:

```
package com.softwareag.cis.util;

public interface IControlStatusConstants
{
    public static final String CS_INVISIBLE = "INVISIBLE";
    public static final String CS_ERROR = "ERROR";
    public static final String CS_FOCUS = "FOCUS";
    ...
}
```

The `Adapter` class from which you inherit your adapters supports this interface - consequently, you can directly use the CS_* constants inside your adapter implementations.

The generic `com.softwareag.cis.server.Adapter` class that you use as the base class for your adapter implementation already implements the `IControlStatusConstants` interface: you can directly use the constant definitions inside your adapter implementation.

The difference in behavior between "FOCUS" and "FOCUS_NO_SELECT" affects only the FIELD and TEXT controls. For these controls, FOCUS set the focus and selects the complete text inside

the control. "FOCUS_NO_SELECT" sets the focus to the control, but does not select the text. For all other controls, "FOCUS_NO_SELECT" behaves like "FOCUS".

For all other controls - and for more complex manipulations of what is visible and not - use the possibility to be able to control the visibility of rows (ITR, TR) or containers (ROWAREA, ROWTABLE0): these controls provide for a visibility property and consequently can be switched on and off.

There is an extended management of what the control status "INVISIBLE" means. Most input controls (FIELD, CHECKBOX, etc.) supporting a `statusprop` or a `visibleprop` also support a property `invisiblemode`. The allowed values of `invisiblemode` are:

- **invisible**
  The corresponding control is completely removed. The horizontal space it occupied before is taken out.
- **cleared**
  The corrresponding control is not visible but still occupies its horizontal space.
- **disabled**
  The corresponding control is displayed with a disabled state. This state is only allowed with a certain number of controls (e.g. button and icon).

Focus Management

Sometimes you want to control the keyboard focus inside a page. Here are the internal rules how a page finds out where to put the focus on.

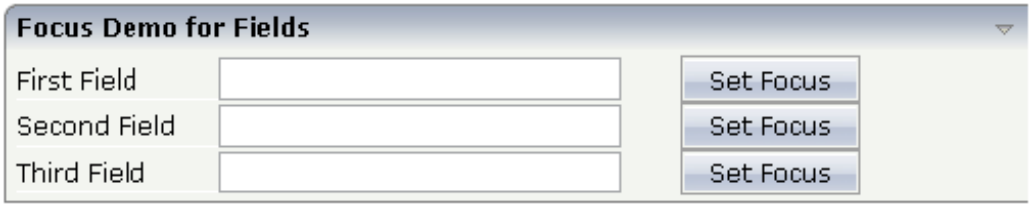
The default reaction is - if a page is displayed for the first time - to put the focus on the first input control (FIELD, CHECKBOX, RADIOBUTTON, etc.) that is available inside a page. After that, you can navigate through the input controls - and the focus is kept stable when interacting with the server.

With `statusprop` - as mentioned in the previous section - you can interrupt this default reaction; there are two possibilities:

- If an input control is set to status "ERROR", it requests the focus automatically. The purpose is to guide the user automatically to those fields that are not correctly entered.
- If an input control is set to status "FOCUS", it is editable - just as normal - and also requests the focus.

If several input controls are requesting the focus at the same time, the focus is put on the first corresponding input control.

A demo page is available in the standard Application Designer workplace showing an example of how to use the focus management:



The XML layout definition for the first area "Focus Demo for Fields" looks as follows:

```
<rowarea name="Focus Demo for Fields">
    <itr>
        <label name="First Field" width="100">
        </label>
        <field valueprop="prop1" width="200" statusprop="prop1status">
        </field>
        <hdist width="30">
        </hdist>
        <button name="Set Focus" method="focus1">
        </button>
    </itr>
    <itr>
        <label name="Second Field" width="100">
        </label>
        <field valueprop="prop2" width="200" statusprop="prop2status">
        </field>
        <hdist width="30">
        </hdist>
        <button name="Set Focus" method="focus2">
        </button>
    </itr>
    <itr>
        <label name="Third Field" width="100">
        </label>
        <field valueprop="prop3" width="200" statusprop="prop3status">
        </field>
        <hdist width="30">
        </hdist>
        <button name="Set Focus" method="focus3">
        </button>
    </itr>
</rowarea>
```

In the demo, each field refers to a status property. The corresponding code of the adapter class looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class FocusManagementAdapter
    extends Adapter
{
    // property >first<
    String m_first;
    public String getFirst() { return m_first; }
    public void setFirst(String value) { m_first = value; }

    // property >second<
    String m_second;
    public String getSecond() { return m_second; }
    public void setSecond(String value) { m_second = value; }

    // property >third<
    String m_third;
    public String getThird() { return m_third; }
    public void setThird(String value) { m_third = value; }

    // property >prop1<
    String m_prop1;
    public String getProp1() { return m_prop1; }
    public void setProp1(String value) { m_prop1 = value; }

    // property >prop1status<
    String m_prop1status;
    public String getProp1status() { return m_prop1status; }
    public void setProp1status(String value) { m_prop1status = value; }

    // property >prop2<
    String m_prop2;
    public String getProp2() { return m_prop2; }
    public void setProp2(String value) { m_prop2 = value; }

    // property >prop2status<
    String m_prop2status;
    public String getProp2status() { return m_prop2status; }
    public void setProp2status(String value) { m_prop2status = value; }

    // property >prop3<
    String m_prop3;
    public String getProp3() { return m_prop3; }
    public void setProp3(String value) { m_prop3 = value; }

    // property >prop3status<
    String m_prop3status;
    public String getProp3status() { return m_prop3status; }
    public void setProp3status(String value) { m_prop3status = value; }
```

```
    public void focus1() { m_prop1status = "FOCUS"; }
    public void focus2() { m_prop2status = "FOCUS"; }
    public void focus3() { m_prop3status = "FOCUS"; }

    /** start of data transfer */
    public void reactOnDataTransferStart()
    {
        super.reactOnDataTransferStart();
        // set default status of fields
        m_prop1status = "EDIT";
        m_prop2status = "EDIT";
        m_prop3status = "EDIT";
    }
}
```

Each time the page communicates with the adapter class, the focus information for all fields is reset to "default" inside the method `reactOnDataTransferStart()`. This method is called prior to any set/get operations by the Application Designer runtime.

By choosing a button, the corresponding `focus` method is called to set the status property to the value "FOCUS".

Flushing of Inputs

Most input controls (FIELD, CHECKBOX, RADIOBUTTON, COMBOFIX, etc.) support a property named `flush`. This property controls whether data input from a user causes an immediate synchronisation with the server or whether data input from a user is stored internally within the client and is synchronized with the next flushing event (e.g. when choosing a button).

There are three different values that can be specified with the `flush` property:

- **""(blank)**
  The data is not synchronized after leaving the control. This is the default.
- **server**
  The data is synchronized with the server immediately when the data has been entered, i.e. when the user has left the corresponding input field.
- **screen**
  The data is synchronized within the controls of the screen. This means - if you have two fields displaying the same property - you can synchronize the fields immediately, without interacting with the server.

Tip: On the one hand, it is useful to flush information in a very fine granular way; you can react on wrong entered data immediately - on the other hand, you have to remember that each flush causes network traffic. The screen's data is sent to the server side processing and

the screen waits for the response of the server. During this time, the page is blocked for input and the user sees an hour glass popping up in the left top corner of the screen.

Tab Sequence

By default, the tab sequence of the controls of a page is defined by the order of the controls inside the page's XML layout definition. Using the property `tabindex`, this order can be overridden and the order of the tab index can be explicitly defined.

The following example shows a page with three fields and one button with an explicitly defined tab sequence:

The XML layout definition is:

```
<rowarea name="Simple Tab Sequence">
    <itr takefullwidth="true">
        <coltable0 width="50%">
            <itr>
                <label name="First" width="120">
                </label>
                <field valueprop="first" width="120" tabindex="1">
                </field>
            </itr>
            <itr>
                <label name="Third" width="120">
                </label>
                <field valueprop="third" width="120" tabindex="3">
                </field>
            </itr>
        </coltable0>
        <coltable0 width="50%">
            <itr>
                <label name="Second" width="120">
                </label>
                <field valueprop="second" width="120" tabindex="2">
                </field>
            </itr>
            <itr>
                <hdist width="120">
                </hdist>
                <button name="OK" method="onOK" tabindex="4">
```

```
                </button>
            </itr>
        </coltable0>
    </itr>
</rowarea>
```

According to the sequence of controls inside the layout definition, the default tab sequence would be: field **First**, field **Third**, field **Second** and button **OK**.

Due to explicitly defining the `tabindex` property for the fields and the button, the tab sequence is now correct: field **First**, field **Second**, field **Third** and button **OK**.

Pay attention:

- Once having started to explicitly set the tab index in a page, you must consequently continue with all controls of the page. Adding new controls without tab index, is internally interpreted as if these controls were defined with tab index "0".
- Equal tab indices in controls are allowed. In this case, the sequence of the controls inside the layout definition defines the tab sequence among the controls with an equal index.
- Moving controls from one location to the other within a page typically means that you have to adapt the tab sequence accordingly.

The tab index usually is a positive integer value. You may define tab index "-1" for excluding certain controls from the tab sequence at all. In this case, the corresponding controls may only be reached by mouse clicking.

Conclusion:

- In typical pages, you do not have to take care of the tab sequence at all because the default (tab sequence by order of controls in page layout) is adequate to the user's experience.
- Only use the explicit definition of the tab sequence if really it is required - the effort for maintaining each tab index with each control should not be underestimated.

Tooltips

Tooltips can be applied to many controls. If the user hovers with the mouse cursor over a control for some seconds, in non-responsive controls, a small yellow box appears showing some more detailed explanation. In responsive controls the tooltips are rendered with more style and stay as long as the mouse cursor is over the control. In responsive controls also icons and arbitrary html can be rendered in tooltips.

The corresponding controls offer the following properties:

- `title`
  Here you can specify a hard-coded text that is used as the tooltip.
- `titletextid`
  Here you specify a text ID that is passed to the multi language management..
- `titleprop`
  Here you can specify a property to dynamically provide the text for the tooltip at runtime. It is mainly supported in responsive controls.
- `titlestraighttext`
  Here you can specify if the text for the tooltip should be rendered as html or as straight text. It is only supported in responsive controls.

22 BREADCRUMB

The BREADCRUMB control represents a horizontal list of method links. The number of links and the name of each link is dynamically derived from the adapter.

The control always occupies 100% of the given width.

Example

The XML layout definition is:

```
<rowarea name="Bread Crumbs...">
    <breadcrumb breadcrumbprop="items">
    </breadcrumb>
</rowarea>
```

The Java code of the adapter is:

```
package com.softwareag.cis.test21;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.BREADCRUMBInfo;
import com.softwareag.cis.server.util.BREADCRUMBItem;

public class BreadcrumbAdapter extends Adapter
{
    // -------------------------------------------------------------------------
    // inner classes
    // -------------------------------------------------------------------------

    // class >MyBCItem<
    public class MyBCItem extends BREADCRUMBItem
    {
        public MyBCItem(BREADCRUMBInfo info, String text)
        {
            super(info, text);
        }

        public void execute()
        {
            Object itemsInfo[] = m_items.getItems();
            MyBCItem items = this;
            m_items.clear();
            m_items = new BREADCRUMBInfo();
```

```
                if (((MyBCItem)itemsInfo[0]).equals(items))
                {
                    new MyBCItem(m_items, "Books");
                }
                else if (((MyBCItem)itemsInfo[1]).equals(items))
                {
                    new MyBCItem(m_items, "Books");
                    new MyBCItem(m_items, "Computers");
                }
                else if (((MyBCItem)itemsInfo[2]).equals(items))
                {
                    new MyBCItem(m_items, "Books");
                    new MyBCItem(m_items, "Computers");
                    new MyBCItem(m_items, "Ajax");
                }
            outputMessage(MT_SUCCESS, "Breadcrumb item: " + getText() + " was ↩
pressed!");
        }
    }

    // -------------------------------------------------------------------------
    // properties
    // -------------------------------------------------------------------------

    // property >infoText<
    String m_infoText;
    public String getInfoText() { return m_infoText; }

    // property >items<
    BREADCRUMBInfo m_items = new BREADCRUMBInfo();
    public BREADCRUMBInfo getItems() { return m_items; }

    // -------------------------------------------------------------------------
    // public usage
    // -------------------------------------------------------------------------

    /**
     */
    public void init()
    {
        m_infoText = "Please click on the breadcrumbs and see the result on the ↩
statusbar";
        new MyBCItem(m_items, "Books");
        new MyBCItem(m_items, "Computers");
        new MyBCItem(m_items, "Ajax");
    }
}
```

The list of items can be changed inside the adapter; the breadcrumb list will react accordingly on the client side. When the user clicks a link on the client side, the `execute()` method will be called in the object of the `BREADCRUMBInfo` collection which represents the link.

Properties

Basic			
breadcrumbprop	Name of the adapter property that represents the control on adapter side. The property must be of name BREADCRUMBInfo. Please view the Java API Documentation for more information.	Obligatory	
breadcrumbstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
pixeldistance	Pixel distance between the links that are rendered.	Optional	1 2 3 int-value
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

23 BUTTON

The BUTTON control represents a button. Within the definition, specify a method that is called in the adapter when choosing the button.

Example: Simple Button

The XML layout definition is:

```
<rowarea name="Buttons">
    <itr>
        <button name="Save As ..." method="saveAs">
        </button>
        <hdist>
        </hdist>
        <button name="Refresh" method="refresh">
        </button>
    </itr>
</rowarea>
```

Example: Button with Image

The XML layout definition is:

```
<rowarea name="Buttons">
    <itr>
       <button name="Save" method="onSave" image="../HTMLBasedGUI/images/save.gif">
        </button>
        <hdist>
        </hdist>
        <button name="Remove" method="onRemove" ↩
image="../HTMLBasedGUI/images/remove.gif">
        </button>
    </itr>
</rowarea>
```

Hiding and Disabling Buttons

Buttons (like many other controls) can be dynamically hidden by using the `visibleprop` property - and referencing to a server side property that decides whether to hide a button or not.

There are two modes of hiding that can be controlled by using the property `invisiblemode`:

- If set to "disabled", the button is grayed and is not selectable anymore.
- If set to "invisible", the button is hidden.

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.	Sometimes obligatory	

	Do not specify a "name" inside the control if specifying a "textid".		
method	Name of the adapter object's method that is called when the user presses the button. Following JAVA coding conventions a method name typically starts with a non-capital character, e.g. "onSave" instead of "ONSave". If no method is specified, the method processAsDefault is called. If the method starts with javascript: the corresponding javascript method is called.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
name	(already explained above)		
textid	(already explained above)		
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. specifying "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	gif jpg jpeg
invisiblemode	This property has three possible values: (1) "invisible": the control is not visible without occupying any space. (2) "disabled": the control is deactivated: it is "grayed" and does not show any roll over effects any more. (3)"cleared": the control is not visible but it still occupies space.	Optional	invisible disabled cleared
switchvisibleproponuserinput	If set to TRUE, the visibleprop is automatically switched to TRUE in case of user input to any input control in this page. The default is FALSE.	Optional	true false

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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
imageheight	Pixel height of image inside button.	Optional	10 20 40

			100 300
imagewidth	Pixel width of image inside button.	Optional	10 20 40 100 300
textstyle	CSS style definition that is directly passed into the text of this control. With the style you can individually influence the text of the button. You can specify any style sheet expressions. Examples are: font-weight: bold color: #FF0000	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
buttonstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
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.	Optional	VAR1 VAR2 VAR3 VAR4

	Purpose: you can set up style variants in the style sheet definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!		
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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
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	Optional	1 2 3

	you may want to define the control to span over more than one columns. The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		4 5 50 int-value
imagedisabled	URL of image that is displayed if the control is disabled. Use properties VISIBLEPROP and INVISIBLEMODE to disable the control.	Optional	gif jpg jpeg
submitbutton	Set this property to true and the button will work as an 'Submitbutton', that is necessary if you want to transfer and/or save form values. i.e. password and username or complete search forms Default value is false. You should only use a 'Submitbutton' if the withformtag option of the pagebody tag is set true.	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
Binding			
method	(already explained above)		
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	

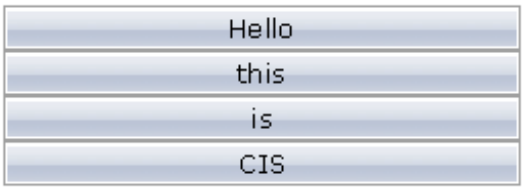
nameprop	Name of adapter property that provides the text to be displayed inside the button. Typically buttons have static texts either defined by the property "name" or "textid". Via "nameprop" you can dynamically set the button's text by your application. Use the nameprop in cases the button's text should change dependent on your logic. Example: you may want to define the button's text to reflect the next status the user can set to a business object.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
imagedisabledprop	Name of adapter property that provides as value the URL of the image that is shown when the control is disabled.	Optional	
focusedprop	Name of property that indicates if the control should receive focus. Must be of type "boolean"/ "Boolean"	Optional	
Online help			
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 language management - representing the tooltip text that is used for the control.	Optional	
titleprop	(already explained above)		
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	

24 BUTTONLIST

The button list represents a vertical arrangement of buttons. The number of buttons and the name on each button are dynamically derived from the adapter.

The controls always occupy 100% of the given width and occupy the height required by the buttons.

Example



The XML layout definition is:

```
<pagebody>
    <buttonlist buttonlistprop="buttonlist">
    </buttonlist>
</pagebody>
```

The Java code of the adapter is:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.BUTTONLISTInfo;
import com.softwareag.cis.server.util.BUTTONLISTItem;

// This class is a generated one.

public class TabSelAdapter
    extends Adapter
{
    // class >MyBLItem<
    public class MyBLItem extends BUTTONLISTItem
    {
        public MyBLItem(BUTTONLISTInfo info, String text)
        {
            super(info, text);
        }

        public void execute()
        {
            outputMessage("S","Button " + getText() + " was pressed!");
        }
    }

    // property >buttonlist<
```

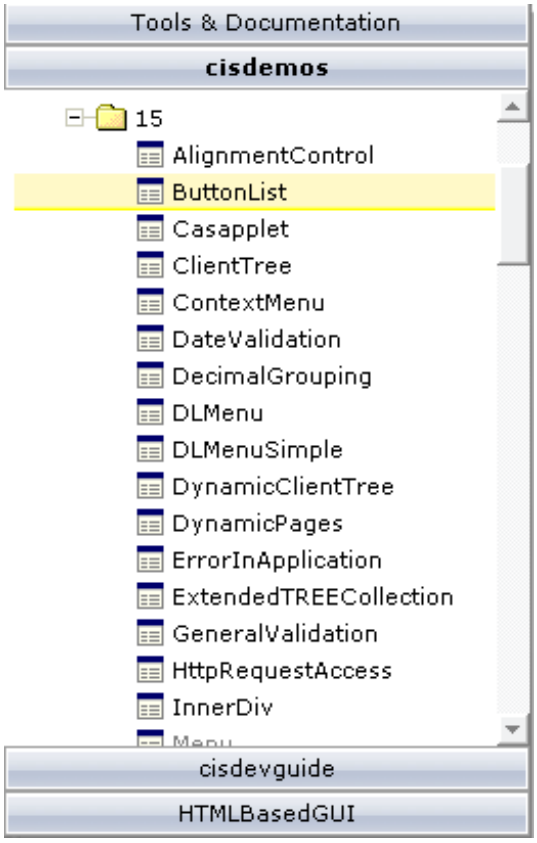
```
    BUTTONLISTInfo m_buttonlist = new BUTTONLISTInfo();
    public BUTTONLISTInfo getButtonlist() { return m_buttonlist; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        // Fill Buttomlist
        MyBLItem item;
        item = new MyBLItem(m_buttonlist,"Hello");
        item = new MyBLItem(m_buttonlist,"this");
        item = new MyBLItem(m_buttonlist,"is");
        item = new MyBLItem(m_buttonlist,"CIS");
    }
}
```

The list of items can be changed inside the adapter. The button list will react accordingly on the client side.

If the user clicks a button on the client side, the `execute()` method will be called in the object of the `BUTTONLISTInfo` collection which represents the button. Your reaction can be added accordingly.

Defining Outlook Bars by Using BUTTONLIST

Sorry for the naming, but this seems to be the most common way to make you understand what is meant: in many applications, you find the following arrangement of controls in order to arrange a set of functions to be called by a user:



The way to build this control is to combine the following:

- One BUTTONLIST at the top of the page.
- One tree control in the middle (see *Working with Trees*).
- One BUTTONLIST at the bottom of the page.

Depending on the clicking of the user, the two BUTTONLIST controls change the number of contained buttons in a way that they “mirror one another”.

Properties

Basic			
buttonlistprop	Name of the adapter property that represents the control on adapter side. The property must be of name BUTTONLISTInfo. Please view the Java API Documentation for more information.	Obligatory	
pixeldistance	Pixel distance between the buttons that are rendered.	Optional	1

			2 3 int-value
buttonstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
imageheight	Pixel height of image inside button.	Optional	10 20 40 100 300
imagewidth	Pixel width of image inside button.	Optional	10 20 40 100 300
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
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
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	

25 CHECKBOX

The CHECKBOX control displays a check box. It represents a boolean value of an adapter property.

Example

The XML layout definition is:

```
<rowarea name="Checkboxes">
    <itr>
        <checkbox valueprop="marked">
        </checkbox>
        <label name="Marked" asplaintext="true">
        </label>
    </itr>
</rowarea>
```

The corresponding Java code of the adapter is:

```
// property >marked<
boolean m_marked;
public boolean getMarked() { return m_marked; }
public void setMarked(boolean value) { m_marked = value; }
```

Properties

Basic			
valueprop	Name of adapter property that is represented by checkbox. The property must be of type "boolean" or "Boolean" (or "String").	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
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	Optional	100 120 140

	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 percentage 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.		160 180 200 50% 100%
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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

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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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
Label			
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.	Optional	
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".	Optional	
hdistpixelwidth	Width of the distance between checkbox and label in pixel.	Optional	

labelstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Binding			
valueprop	(already explained above)		
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server

flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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 language management - representing the tooltip text that is used for the control.	Optional	
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	

Typically, the CHECKBOX is followed by a LABEL control naming the displayed check box. In the LABEL definition, set the property `asplaintext` to "true".

26 COMBODYN2

The COMBODYN control is the dynamic counterpart of the COMBOFIX control. Whereas the selection options inside the COMBOFIX control are defined in a fixed way inside the page definition, the COMBODYN2 control offers the possibility to derive the selection options dynamically from adapter properties.

Example

The XML layout definition looks as follows:

```
<rowarea name="ComboDyn">
    <itr>
        <label name="Cost Center" width="120">
        </label>
        <combodyn2 valueprop="costCenter" validvaluesprop="validCostCenters"
                   width="200" size="1">
        </combodyn2>
    </itr>
</rowarea>
```

The definition of the COMBODYN2 control refers to a `valueprop` property: this is the property of the adapter class in which the selection is actually passed. In addition, the definition refers to a `validvaluesprop` property: this is the property from which the options are taken.

The code of the corresponding adapter class looks as follows:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.COMBODYNValidValues;

// This class is a generated one.

public class ComboFixAdapter
    extends Adapter
{
    // property >costCenter<
    String m_costCenter;
    public String getCostCenter() { return m_costCenter; }
    public void setCostCenter(String value) { m_costCenter = value; }

    // property >validCostCenters<
    COMBODYNValidValues m_validCostCenters = new COMBODYNValidValues();
    public COMBODYNValidValues getValidCostCenters() { return m_validCostCenters; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_validCostCenters.addValidValue("0001","Marketing");
```

```
        m_validCostCenters.addValidValue("0002","Sales");
        m_validCostCenters.addValidValue("0003","Development");
    }
}
```

Typical Problems with COMBODYN2

The rendering problems with the internally used HTML control SELECT also apply for the COMBODYN2 control. See the corresponding information in the section *Typical Problems with COMBOFIX*.

For this reason, COMBODYN2 offers the property `renderasfield`: when switched to "true", the rendering is not done by using the HTML control SELECT, but by using the normal Application Designer FIELD with valid value support. Rendering as FIELD has the following advantages:

- There are no overlapping conflicts anymore.
- Valid values are brought to the client at the point of time when the user requests value help.

But there is also a disadvantage:

- When selecting a value from the valid value list, the value is displayed with its ID - not with its description.

Properties

Basic			
valueprop	Server side property representation of the control.	Obligatory	
validvaluesprop	Adapter property that provides for the valid values that are available as selectable options. The adapter property must be of type "COMBODYNValidValues".	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").	Sometimes obligatory	100 120 140 160 180 200

	(C) Percentage sizing: input a percentage 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.		50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
size	Number of rows that are displayed inside the control. If specified as "1" (default) then the control is rendered as combo box - if ">1" then the control is rendered as multi line selection.	Optional	
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	
direction	Presets the default(BiDi) direction of the control. Use black string in order to have the default value.	Optional	rtl ltr
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom

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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
renderasfield	If set to "true" then the combo box is rendered like a FIELD control that offers valid value support. Default is "false". The normal translation of COMBODYN2 into HTML renders an HTML-select control. This control has certain limitations inside Internet Explorer: it only offers a very reduced set of styles to manipulate its look and feel and - much worse: it always occupies z-index "0" i.e. if you other areas overlapping the COMBODYN2 area then COMBODYN2 is always on the top. This is quite ugly if e.g. a menu is opened and parts of the menu overlap a COMBODYN2 control.	Optional	true false
allowmultiselection	If set to true then multiple selections are allowed.	Optional	true false
combostyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:	Optional	

	border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	xs:string ---------- N n.n P n.n string n
Binding			
valueprop	(already explained above)		
validvaluesprop	(already explained above)		

displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
title	Text that is shown as tooltip for the control.	Optional	

	Either specify the text "hard" by using this TITLE property - or use the TITLETEXTID in order to define a language dependent literal.		
titleprop	(already explained above)		
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	

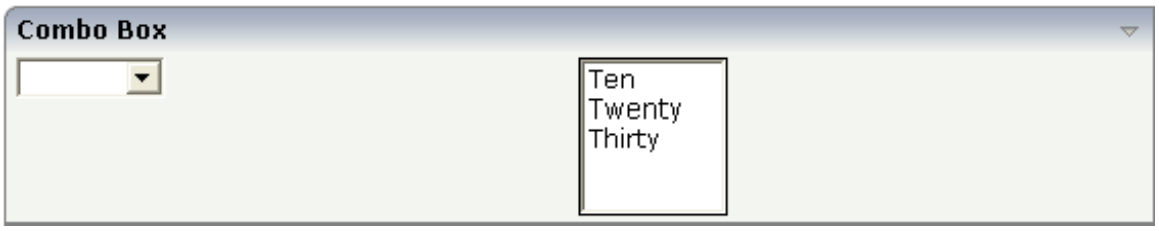
27 COMBOFIX

The COMBOFIX control is a selection control. Depending on its configuration, it is either displayed as a combo box or as a selection list.

The COMBOFIX control allows specifying a defined set of values which can be selected. This set of values is defined as part of the layout definition - it cannot be loaded dynamically from the server.

Note: If you want to use dynamic selection, there are two possibilities. Either use the COMBODYN control which has the same look and feel as the COMBOFIX control, but where the selectable values are not specified as part of the page definition and are derived from an adapter property. Or use the value help pop-up dialogs as described in *Working with Page Navigation*.

Example



The XML layout code for the example looks as follows:

```
<rowarea name="Combo Box">
    <itr takefullwidth="true">
        <coltable0 width="50%" takefullheight="true">
            <itr>
                <combofix valueprop="factor1" size="1" flush="screen">
                    <combooption name="Ten" value="10">
                    </combooption>
                    <combooption name="Twenty" value="20">
                    </combooption>
                    <combooption name="Thirty" value="30">
                    </combooption>
                </combofix>
            </itr>
        </coltable0>
        <coltable0 width="50%" takefullheight="true">
            <itr takefullwidth="false">
                <combofix valueprop="factor1" size="5" flush="screen">
                    <combooption name="Ten" value="10">
                    </combooption>
                    <combooption name="Twenty" value="20">
                    </combooption>
                    <combooption name="Thirty" value="30">
                    </combooption>
                </combofix>
            </itr>
        </coltable0>
    </itr>
</rowarea>
```

There is a property `size` inside the COMBOFIX definition. This property specifies the number of lines that are displayed inside the control. If the `size` property is set to "1", a combo box is displayed; if it is set to a value higher than "1", the selection control is displayed.

Inside the COMBOFIX definition, there is a set of COMBOOPTION definitions that represent the selectable values. Each value consists of a display string (`name` property) and a value that is actually set as a property value (`value` property).

The COMBOFIX definition contains a reference to an adapter property (`valueprop` property) into which the value is transferred.

Typical Problems with COMBOFIX

The COMBOFIX control is internally rendered as the HTML control SELECT. Unfortunately, the HTML control has some internal problems and behaves different from normal HTML controls:

- The HTML control SELECT always stays on top of all controls, i.e. it has a “maximum high” z-index and does not allow other controls to overlap.
- The HTML control SELECT does not allow sophisticated style definitions: besides some coloring aspects, you have to accept the look and feel of this control. For example, you cannot turn off its thick borders and you cannot influence the selection image which shows valid values.

Be aware of the consequences of these problems when using the control. Do not place the control into a page are in which a menu might overlap. The COMBOFIX will always be on top of the menu.

COMBOFIX Properties

Basic			
valueprop	Server side property representation of the control.	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 percentage 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%

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
size	Number of rows that are displayed inside the control. If specified as "1" (default) then the control is rendered as combo box - if ">1" then the control is rendered as multi line selection.	Optional	
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	
direction	Presets the default(BiDi) direction of the control. Use black string in order to have the default value.	Optional	rtl ltr
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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

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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
combostyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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

datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	xs:string ---------- N n.n P n.n string n
Binding			
valueprop	(already explained above)		
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			

helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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	
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	

COMBOOPTION Properties

Basic			
name	Name that is displayed as selectable option. Either use the NAME property to specify the text in a "hard" way or use the TEXTID property to define the text in a language dependent way.	Optional	
textid	Text ID that is used for this option. The text id is passed to the multi language management in order to find a language dependent text.	Optional	
value	Actual value of the option that is passed into the adapter property specified by VALUEPROP inside the COMBOFIX control.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

28 DATEINPUT

The DATEINPUT control is used to input a date or a date with time. The input can be done both with the keyboard or by opening a pop-up in which the user can browse through a calendar. The calendar can be controlled by server side processing in the following way:

- You can define a valid-from and a valid-to date. Thus, the control will not allow the user to input an invalid date.
- You can explicitly control the color and the tooltip information inside the calendar. For example, you may set up a calendar in which vacation times are highlighted in a certain way.

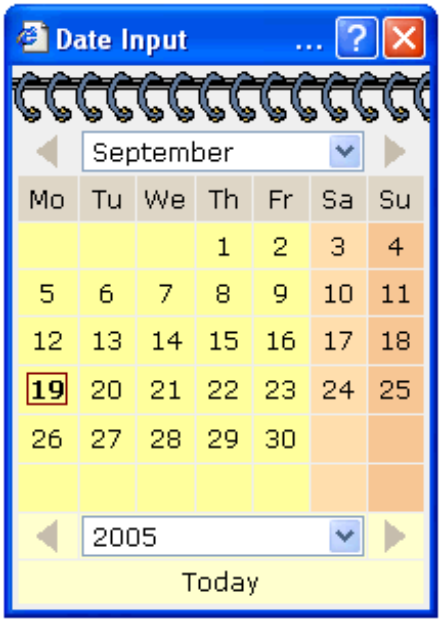
Example

The most simple usage scenario is to just use the DATEINPUT control in the following way:

```
<rowarea name="Dateinput">
    <itr>
        <label name="Order Date" width="120">
        </label>
        <dateinput valueprop="orderDate" width="120">
        </dateinput>
    </itr>
</rowarea>
```

The corresponding screen looks like this:

The user may directly enter a date into the field (the format of the input is specified by the session's date display settings) or may open the calendar pop-up by clicking on the field's icon:



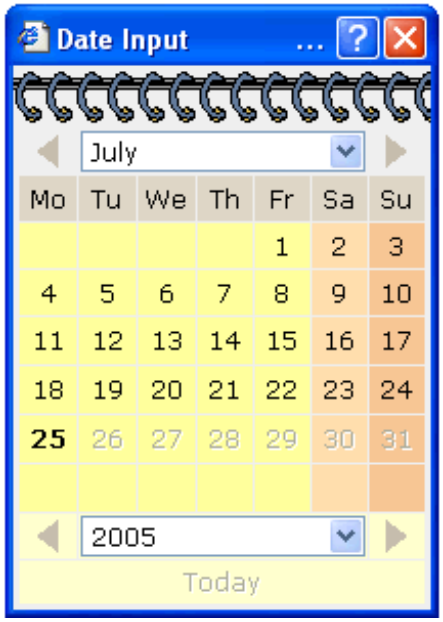
The server-side adapter class looks like this:

```
public class DateInputAdapter
    extends Adapter
{
    // property >orderDate<
    CDate m_orderDate;
    public CDate getOrderDate() { return m_orderDate; }
    public void setOrderDate(CDate value) { m_orderDate = value; }
}
```

Instead of using a property of type `CDate`, you may also use a simple `String` property. In this case, the date string is passed in the format "YYYYMMDD".

From-To Restrictions

By using the control's properties `fromprop` and `toprop`, you can bind the control to adapter properties that define a time range that can be selected by the user:



In the date control, you can only input dates between the 3rd and the 15th of June. The corresponding tag definition of the DATEINPUT control looks like this:

```
<dateinput valueprop="orderDate" width="120" fromprop="fromDate" toprop="toDate">
</dateinput>
```

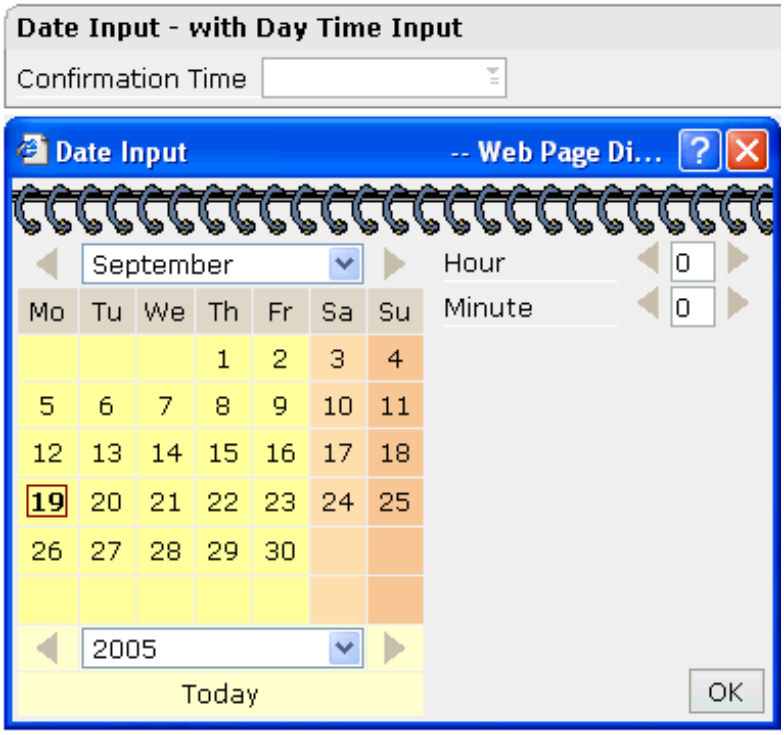
The server-side adapter holds the properties `fromDate` and `toDate`:

```
public class DateInputAdapter
    extends Adapter
{
    // property >orderDate<
    CDate m_orderDate;
    public CDate getOrderDate() { return m_orderDate; }
    public void setOrderDate(CDate value) { m_orderDate = value; }

    // property >fromDate<
    CDate m_fromDate = new CDate("20050701");
    public CDate getFromDate() { return m_fromDate; }
    public void setFromDate(CDate value) { m_fromDate = value; }

    // property >toDate<
    CDate m_toDate = new CDate("20050725");
    public CDate getToDate() { return m_toDate; }
    public void setToDate(CDate value) { m_toDate = value; }

}
```

You may also specify only the `fromprop` property: in this case, the user may select all dates following the specified date. (Same with only specifying the `toprpop` property.)

Input of Date and Time

By using the property `datatype`, you can use the DATEINPUT control for inputting a time stamp:



The corresponding layout definition is:

```
<rowarea name="Date Input - with Day Time Input">
    <itr>
        <label name="Confirmation Time" width="120">
        </label>
        <dateinput valueprop="confirmationTime" width="120" datatype="datetime">
        </dateinput>
    </itr>
</rowarea>
```

The control now refers on the server side to a property of type `CTimestamp`:

```
// property >confirmationTime<
CTimeStamp m_confirmationTime;
public CTimeStamp getConfirmationTime() { return m_confirmationTime; }
public void setConfirmationTime(CTimeStamp value) { m_confirmationTime = value; }
```

Using the property `secondsvisprop`, you can bind the control to a boolean adapter property which decides whether the user can input the time with or without seconds.

Properties

Basic			
valueprop	Server side property representation of the 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 percentage 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%
datatype	By default, the DATEINPUT control is managing a day. By explicitly setting a datatype you can define that the control is managing a day and time. In the first use type CDATE within your adapter program - in the second case use type CTIMESTAMP.	Optional	date datetime ---------- xs:date xs:dateTime
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)		
fromprop	Name of an adapter property to provide for a lower limit. Property must return an object of type CDATE. The value is used for client side validation of user input.	Optional	
toprop	Name of an adapter property to provide for a upper limit. Property must return an object of type CDATE. The value is used for client side validation of user input.	Optional	
infoprop	Name of an adapter property to provide for style information that is used inside the date popup. The property must return an object of type DATEINPUTInfo.	Optional	
secondsvisprop	Name of an adapter property to provide for a boolean that indicates if to show additional seconds. This property make sense only if property DATATYPE is set to "daytime".	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the	Optional	

	content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.		
Appearance			
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
inputstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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 synchronized 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
noborder	Boolean value defining if the control has a border. Default is "false".	Optional	true false
transparentbackground	Boolean value defining if the control is rendered with a transparent background. 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
Valuehelp			
popupicon	URL of image that is displayed inside the right corner of the field to indicate to the user that there is some value help available.. 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. specifying "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	gif jpg jpeg
popupinputonly	Boolean property that control if a field with POPUPMETHOD defined is still usable for keyboard input. If "false" (= default) then the user can input a value either directly via keyboard or by using the popupmethod's help. If set to "true" then no keyboard input is possible - but only selection from the popup-method's help.	Optional	true false
popuponalt40	Value help in a field is triggered either by clicking with the mouse or by pressing a certain key inside the field. The "traditional" keys are "cursor-down", "F7" or "F4". Sometimes you do not want to mix other "cursor-down" behaviour (e.g. scrolling in lists) with the value help behaviour. In this case switch this property to "true" - and the value help will only come up anymore when "alt-cursor-down" is pressed.	Optional	true false
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	

helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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	

29 DATEINPUT2

The DATEINPUT2 control supports the selection and input of a date from a calendar. It has a more modern look and feel than the DATEINPUT control and does not require browser pop-ups.

Customizing Date and Calendar Formats

Default behavior for the date and calendar formats for NATPAGE layouts:

- **Date format in browser:**
  Per default the dates are shown in the browser according to the Natural DTFORM parameter.
- **Date format in Natural program:**
  Depending on the `datatype` property the Natural type D or an (A/U) DYNAMIC type is used. For the latter the format on the server is YYYYMMDD.
- **Calendar format:**
  The first day in week of the calendar is per default Sunday and can be customized per application and per single page at runtime

Some use cases require the formats to be customized independently of the central format at design time and in a more flexible way. This is supported by the properties `clientformat`, `severformat`, and `firstdayinweek`. If these properties are set in a control, they will overwrite the default behavior. For the `clientformat` and the `serverformat` property a subset of the Natural date edit masks is supported:

Character	Usage
DD	Day
ZD	Day with zero suppression
MM	Month
ZM	Month with zero suppression
YYYY	Year, 4 digits
YY	Year, 2 digits

DATEINPUT2 – Custom Dates

Per default, the DATEINPUT2 control validates inserted dates according to a range of valid dates. Inserting dates outside of this range are not accepted by the user interface. Sometimes applications use predefined date values to reflect an application-specific semantic meaning. Example: An application may use the date “9999-99-99” to express a never expiring license. For these applications, an end-user must be able to enter these specific custom dates without getting validation errors.

This is supported for DATEINPUT2 controls for which a `clientformat` is set. The custom dates can be specified as the configuration setting `customdates` in the *cisconfig.xml* configuration file.

Example:

```
<cisconfig customdates="99999999;99991231" ….></cisconfig>
```

In the `customdates` property, you can specify several `customdates` separated by a semi-colon. The format for these custom dates is `YYYYMMDD`. When a custom date is entered in the browser no validation error is displayed. They are rendered according to the `clientformat` property value and passed to the server according to the `serverformat` property value if specified.

Example:

```
<dateinput2 clientformat="YYYY-MM-DD" serverformat="DD/MM/YYYY"…>
```

Typing „99999999“ in the browser will be rendered as “9999-99-99” and sent to the server as “99/99/9999”.

For custom dates, the calendar widget is not available. It appears again once the custom date is removed or changed to a “normal date”.

Properties

Basic			
valueprop	Server side property representation of the 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 percentage 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	Optional	100 120 140 160 180 200 50% 100%

	width of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		
datatype	By default, the DATEINPUT control is managing a day. By explicitly setting a datatype you can define that the control is managing a day and time. In the first use type CDATE within your adapter program - in the second case use type CTIMESTAMP.	Optional	date datetime --- xs:date xs:dateTime
serverformat	For alphanumeric datatypes you can choose the format of the date at design time. Examples: YYYY-MM-DD, DD.MM.YY, YY/MM/DD. A subset of the Natural date edit masks is supported. The serverforma is the format in which the data is sent to Natural.	Optional	YYYY-MM-DD DD/MM/YYYY MM-DD-YY
clientformat	You can choose the format of the date in the browser at design time. Examples: YYYY-MM-DD, DD.MM.YY, YY/MM/DD. A subset of the Natural date edit masks is supported. If set the Natural DTFORM parameter will not be used. This setting is only for the rendering in the client. It is independent of the set datatype.	Optional	YYYY-MM-DD DD/MM/YYYY MM-DD-YY
firstdayinweek	You can set the first day in week at design time. Valid values are SU - for Sunday - and MO - for Monday. If set the value set by the NJX:SESSIONPARAMS control will not be used.	Optional	SU MO
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)		
fromprop	Name of an adapter property to provide for a lower limit. Property must return an object of type CDATE. The value is used for client side validation of user input.	Optional	
toprop	Name of an adapter property to provide for a upper limit. Property must return an object of type CDATE. The value is used for client side validation of user input.	Optional	
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false).	Optional	

	By using this property you can dynamically control the "display"-status of the control by your adapter object.		
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
holidaysurlprop	Name of the Adapter property which provides the URL for a json file with custom holidays dynamically at runtime.	Optional	
Appearance			
popupicon	URL of image that is displayed inside the right corner of the field to indicate to the user that there is some value help available.. Any image type (.gif, .jpg, ...) that your browser does understand is valid. Use the following options to specify the URL:	Optional	gif jpg jpeg

	(A) Define the URL relative to your page. Your page is generated directly into your project's folder. specifying "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".		
popuponalt40	Value help in a field is triggered either by clicking with the mouse or by pressing a certain key inside the field. The "traditional" keys are "cursor-down", "F7" or "F4". Sometimes you do not want to mix other "cursor-down" behaviour (e.g. scrolling in lists) with the value help behaviour. In this case switch this property to "true" - and the value help will only come up anymore when "alt-cursor-down" is pressed.	Optional	true false
popuponF4F7	Per default the calendar is opened on F4 and F7. Set this property to false if you want to use F4 or F7 for other purpose.	Optional	true false
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
numberofmonths	Number of months shown for selection. Default is 1	Optional	
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control.	Optional	left center right

	If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.		
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
inputstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2
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	Optional	1 2 3

	you may want to define the control to span over more than one columns. The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		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 synchronized 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
noborder	Boolean value defining if the control has a border. Default is "false".	Optional	true false
transparentbackground	Boolean value defining if the control is rendered with a transparent background. 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
holidaysurl	URL for json file, which contains custom holidays.	Optional	
holidaysstyleclass	Name of the css style class, which is used for the rendering of custom holidays. Default is DATEINPUT2Holidays	Optional	
holidaysdescriptionastooltip	Set this property to true if you want to show descriptions in the json file as tool tips.	Optional	true

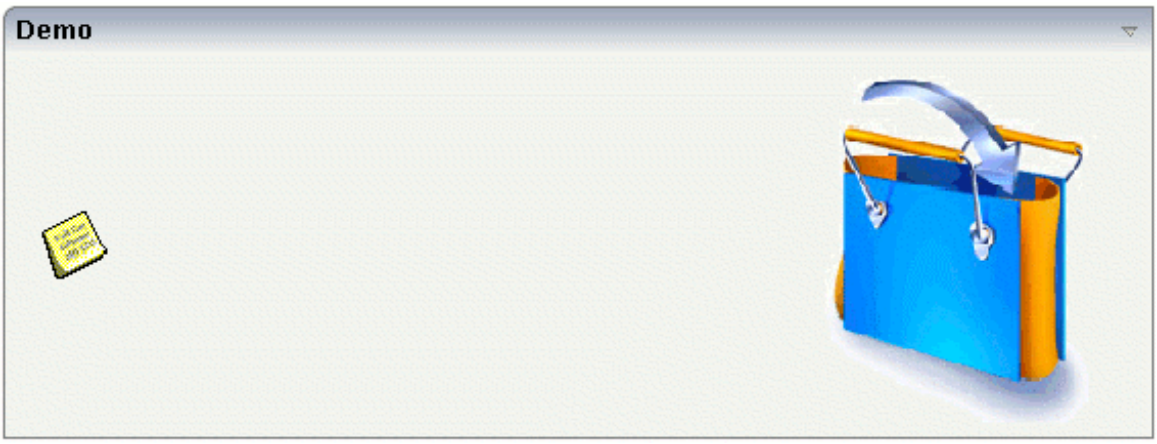
			false
Online Help			
title	Text that is shown as tooltip for the control. Either specify the text "hard" by using this TITLE property - or use the TITLETEXTID in order to define a language dependent literal.	Optional	
titletextid	Text ID that is passed to the multi language management - representing the tooltip text that is used for the control.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
placeholder	The text for the HTML placeholder attribute. The placeholder attribute specifies a short hint that describes the expected value.	Optional	
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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	

30 DROPICON

The DROPICON control is an icon that can be used in order to build drag-and-drop scenarios. A DROPICON can be defined as the starting point of a drag-and-drop operation or as the target point of a drag-and-drop operation.

Example

Have a look at the following screen:



The user can click the left mouse button on the left icon (drag), move the mouse to the right icon and then release the mouse button (drop).

The configuration of drag and drop is quite simple: the icon that is used for starting drag-and-drop operations leaves a certain drag information - a plain string object. The receiving icon, on which the user performs the drop operation, receives both an event and the string object which was left by the icon from where the operation was started.

Have a look at the XML layout definition of the example above:

```
<rowarea name="Demo">
    <itr takefullwidth="true">
        <hdist width="10">
        </hdist>
        <dropicon image="../_DevelopersGuide/images/fav_notes.gif"
                  method="onSomeOtherFunctionality"
                  draginfo="Icon NOTICES has been dropped" dropmethod="onDrop"
                  dropinfoprop="onDropInfo">
        </dropicon>
        <hdist width="100%">
        </hdist>
        <dropicon image="../_DevelopersGuide/images/cishop.gif"
                  draginfo="HAND BAG has been dropped" dropmethod="onDrop"
                  dropinfoprop="onDropInfo">
```

```
        </dropicon>
        <hdist width="10">
        </hdist>
    </itr>
</rowarea>
```

In the left icon, the string that is used as drag information is defined using the `draginfo` property. (There is also a `draginfoprop` property that allows to specify this information using an adapter property instead of hard-wiring it in the layout.) You see that the icon still has a normal `method` property, i.e. it can still be used as a normal icon - invoking a certain adapter method when being chosen.

In the right icon, the `dropmethod` property defines the method that is called inside the adapter when the user drops certain information onto this icon. The property `dropinfoprop` defines the adapter property in which the string representing the drag information is written.

A DROPICON can be both defined as the starting point and as the target point of a drag-and-drop operation.

Dragging and Dropping Information from DROPICON to TREENODE3

The management of trees is described in *Working with Trees*. It is rcommended that you first read the general information about building trees with TREENODE3 there.

In a tree, each tree node is represented by a node object which is an extension of the `NODEInfo` class. If the user drops information from a DROPICON onto a tree node, then the method `reactOnDropGeneric` is called inside the node object. By calling the method `getDragInfo()`, you receive the drag information that was dropped onto the tree node.

Dragging and Dropping Information from DROPICON to ICONLIST

Each ICONLIST item is represented by an object of type `ICONLISTItem`. This object provides a method `reactOnDropGeneric()` which is called when information is dropped, and it provides a method `getDragInfo()` for accessing the dropped information.

Properties

Basic			
image	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Obligatory	gif jpg jpeg
draginfo	String containing any kind of application data to identify the source DROPINFO control within a drag and drop process. Use property DROPINFOPROP to return this data on runtime.	Optional	
draginfoprop	Name of an adapter property that provides for information that is passed to the adapter when dropping this control over another DROPICON. Do not use this property (or property DROPINFO respectively) if you do not want the user to drag this control. The server side property needs to be of type "String".	Optional	
dropinfoprop	Name of an adapter property to that the "drag info" of the dragged DROPICON control is set. Do not use this property if this control should not accept other DROPICON controls within a drag and drop process (i.e. is not a drop target). The server side property needs to be of type "String".	Optional	
dropmethod	Method of your adapter object that is executed when the user is dragging another DROPICON control over this control and drops it there. Do not use this attribute if this control should not accept other DROPICON controls within a drag and drop process (i.e. is not a drop target).	Sometimes obligatory	
method	Method of your adapter object that is executed when clicking on the control.	Sometimes obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
draginfoprop	(already explained above)		
dropinfoprop	(already explained above)		
dropmethod	(already explained above)		

imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
method	(already explained above)		
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
Appearance			
image	(already explained above)		
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
imageinactive	If the visibility is dynamically controlled by using the INVISIBLEPROP then there are two ways the icon reacts if the corresponding property passes back "false". If you want the icon to switch into an inactive status then define inside this property the URL of the image that is the inactive counter part to the normal icon image. Maybe the image is a grayed version of the normal icon image. If you do not define a value for this property then the icon is made invisible.	Optional	
imagewidth	Pixel width of the image that is shown inside the icon. If not defined then the icon is rendered with its normal width.	Optional	
imageheight	Pixel height of the image that is shown inside the icon. If not defined then the icon is rendered with its normal height.	Optional	
withdistance	If set to "true" then 2 pixels of distance are kept on the left and on the right of the icon. Reason behind: if arranging several icons inside one table row (ITR, TR) then a certain distance is kept between the icons when this property is set to "true".	Optional	true false
align	Horizontal alignment of control in its column.	Optional	left center

	Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.		right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
colstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
spanstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
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
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	
titleprop	(already explained above)		

31 FIELD

The FIELD control is used for entering data. It provides the following features:

- Normal input/output of text.
- Password input.
- Autocomplete (see also the AUTOCOMPLETE control).
- Dynamic control if input is allowed.
- Dynamic highlighting of field in case of errors.
- Flush the input directly to the server when leaving the field.
- Start a server method on pressing F4 or F7 or on click - useful for value help pop-up dialogs
- Adapt the output to a data type (e.g. transfer "YYYYMMDD" to a visible date field)

Example

The XML layout definition is:

```
<rowarea name="Fields">
    <itr>
        <field valueprop="factor4" flush="screen" length="10">
        </field>
        <hdist>
        </hdist>
        <field valueprop="factor4" flush="screen" length="10" password="true">
        </field>
        <hdist>
        </hdist>
        <field valueprop="factor4" flush="screen" length="10" displayonly="true">
        </field>
        <hdist>
        </hdist>
        <field valueprop="factor4" flush="screen" length="10">
        </field>
        <hdist>
        </hdist>
    </itr>
</rowarea>
```

For better visibility, distance controls were added between the FIELD controls.

Dynamically Defining the Input Status

As mentioned previously, you can dynamically control the input status of a FIELD by a property of the adapter class. The following example shows how to do this.

The XML layout looks as follows:

```
<rowarea name="Dynamic Field">
    <itr>
        <field valueprop="factor1" flush="server" length="10">
        </field>
        <hdist>
        </hdist>
        <field valueprop="factor1" flush="server" statusprop="factor1status" ↩
length="10">
        </field>
    </itr>
</rowarea>
```

There are two fields that show the same adapter property `factor1`. The first field definition is without any restrictions, the second one depends on the input status of the property `factor1Status`.

The adapter program looks as follows:

```
// property >factor1<
int m_factor1=5;
public int getFactor1() {return m_factor1;}
public void setFactor1(int value) { m_factor1 = value; }

//  property >factor1status<
String m_factor1status;
public String getFactor1status()
{
    if (m_factor1 > 100)     return "DISPLAY";
    else if (m_factor1 > 10)    return "ERROR";
    else                    return "EDIT";
}
public void setFactor1status(String value) { m_factor1status = value; }
```

Now let us see what happens if different numbers are entered:

The right field changes its input status according to the value of the property `factor1Status`. There are four different values that can be returned as status information:

- **EDIT**
  The field is displayed as a normal field.
- **ERROR**
  The field indicates an error with its value.
- **DISPLAY**
  The field is only displayed.
- **FOCUS**
  The field is displayed as a normal field; it requests the focus.

Client Side Validation

By using regular expressions, you can check the user's input into a field in a very powerful way. Regular expressions are a standardized way (W3C) of describing the format of strings. You can use it, for example, to check whether the user entered an article number correctly - following some conventions that are defined inside your application.

Regular expressions can be plugged to a field control so that it checks the input of the user against the expression. The check is done when the user has left the field. If the check is successful, nothing happens - if it fails, an error message pops up indicating to the user that the input did not match the field's requirements.

There are two ways of plugging regular expressions to the field:

- **Static Definition**
  The regular expression is directly defined inside the control's definition.
- **Dynamic Definition**
  Inside the control, you specify a property that passes the expression at runtime.

The following example shows a field in which you enter a telephone number. A regular expression checks whether the number is entered in the right format:

The field is defined in the following way:

```
<field valueprop="phone" width="100" validation="[0-9 )(-/+]+">
</field>
```

The regular expression "[0-9)(-/+]+" indicates that the following can be entered

- A string that has any number of characters: "[]+".
- A string that has only characters which are "0-9)(-/+".

If the user enter a wrong value, the following message appears:

Decimal Number Input

In the field, you can specify the float value for the `datatype` property. Consequently, the field's input will automatically be interpreted as float input - and e.g. decimal separators will be added.

In addition, you can specify the number of valid decimal digits: the number is not defined in a fixed way inside the control but is derived from a server side property (`decimaldigitsprop` property). Maybe you have an application that inputs and outputs amounts with a certain currency reference. Depending on the currency, the number of decimal digits behind the comma may be different.

Value Help

The FIELD control supports a value help - i.e. you can offer the user a support pop-up for a field that e.g. lets the user select valid values instead of typing them manually. The value help bases on a generic mechanism that allows you to define any kind of your own value help pop-ups - but there are also two predefined ways to quickly create a simple value help that lets the user select values from a list.

First, the description of the generic framework: The FIELD control has a property `popupmethod`. If you fill this property, then there are two consequences:

- The field shows a little icon on the very right.
- The field is value help-sensitive: if the user clicks on the icon or clicks with the right mouse button into the field, then the method on server side that is referenced by the `popupmethod` property is called.

You see that the value help is triggered in the client, but the actual value help processing is launched from the server adapter method that is referenced. What the method does is completely up to you - in most cases, it shows a certain pop-up.

Have a look at the following example:

The XML layout definition is:

```
<rowarea name="Address">
    <itr>
        <label name="Titel" width="120">
        </label>
        <field valueprop="titel" width="50" popupmethod="openIdValueHelp">
        </field>
    </itr>
</rowarea>
```

The implementation in the adapter is:

```
// property >titel<
String m_titel;
public String getTitel() { return m_titel; }
public void setTitel(String value) { m_titel = value; }

public void onValueHelpTitel()
{
   openPopup("/HTMLBasedGUI/empty.html");
}
```

When the user chooses the icon in the title field, `onValueHelpTitle()` is called. The method itself opens a certain pop-up. It is completely up to you to specify the reaction - maybe you do not want to open a pop-up but want to navigate to another page.

See *Working with Page Navigation* in the *Java Page Layout > Working with Pages* documentation for more details on pop-up management. Be aware of the fact that - just as with any other method which is, for example, called by a button - all the data of the screen is first transferred into your

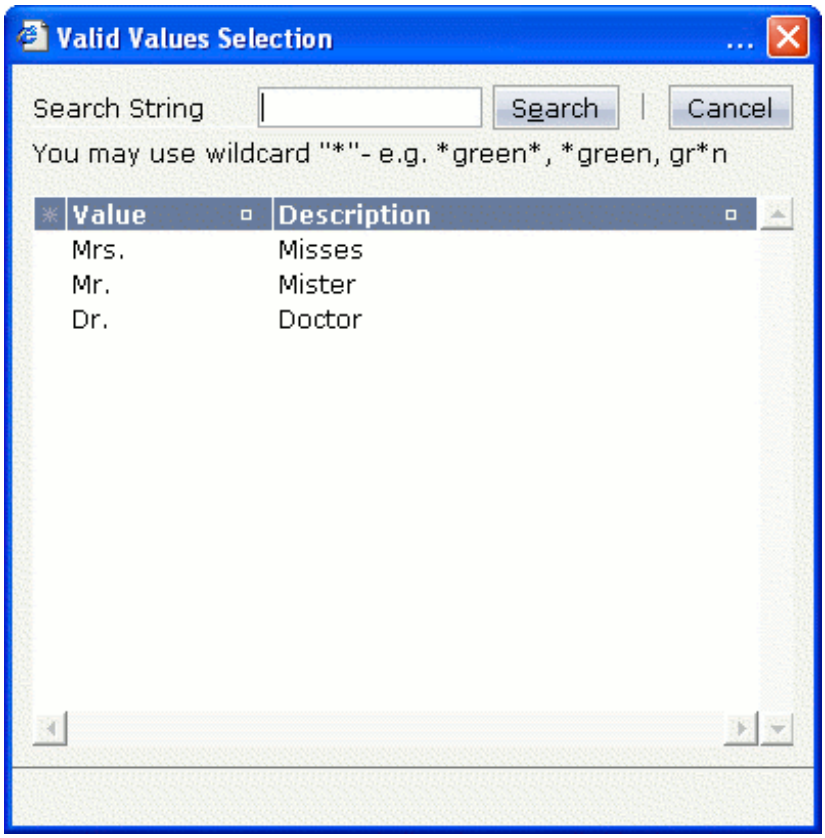
adapter before the method is called. For example, if the user enters "M" into the title field and then invokes the value help, then `setTitle()` is invoked first and after this `onValueHelpTitle()` is invoked. The same happens to any other data that was modified on the screen prior to the help request.

Sometimes you want to define a generic way of reacting to value requests - you do not want to have one explicit method per field to be called - but you want to define one method referenced by all fields. For this purpose, there is a method `findValueRequestProperty()` that you inherit from the Adapter class. This method returns the name of the property that is referenced as `valueprop` inside the corresponding field.

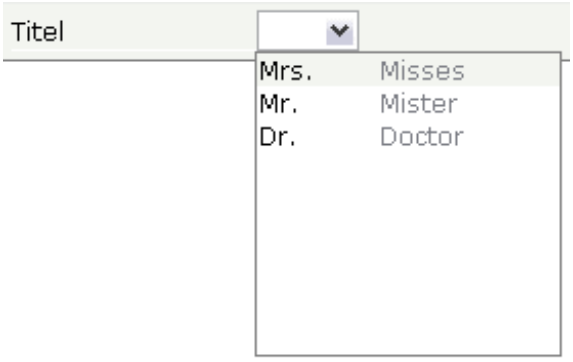
Value Help - Predefined Reaction Methods

Based on the `popupmethod` mechanism that is explained in the previous section, there are simple ways of providing a standard value help for field inputs:

- The predefined pop-up method `openIdValueHelp` requests a list of valid values from the adapter and displays the list in a pop-up from which the user can select a value. The list is fetched by following a certain naming convention: the adapter must provide for a method with the name `findValidValuesForXxx()` where "Xxx" is the name of the property.



- The predefined pop-up method `openIdValueCombo` uses the same `findValidValuesForXxx()` method, but displays the result similar to a combo box:



- The predefined pop-up method `openIdValueComboOrPopup` is a mixture of the methods described above. For performance reasons, small lists are displayed in a combo box and large lists are displayed in a pop-up. By default, lists containing up to 100 entries are shown in a combo box. Using the parameter `maxitemsinfieldcombo` of the configuration file *cisconfig.xml*, you can control the maximum number of entries that are to be shown in the combo box.

The corresponding adapter code is:

```
// property >titel<
String m_titel;
public String getTitel() { return m_titel; }
public void setTitel(String value) { m_titel = value; }

public ValidValueLine[] findValidValuesForTitel()
{
    ValidValueLine[] result = new ValidValueLine[3];
    result[0] = new ValidValueLine("Mrs.","Misses");
    result[1] = new ValidValueLine("Mr.","Mister");
    result[2] = new ValidValueLine("Dr.","Doctor");
    return result;
}
```

Note that the method is called at the point of time when the user requests value help.

The `ValidValueLine` also supports a constructor in which only the value of the field is passed - without further description.

Input-Sensitive Value Help

When having read the previous sections on value help, be aware that at the point in time when the value help is called (e.g. when `findValidValuesFor...()` is called), all data input that was done on the browser client has already been transferred into your adapter object.

This means: you already have access to the property that a user entered before invoking the value help.

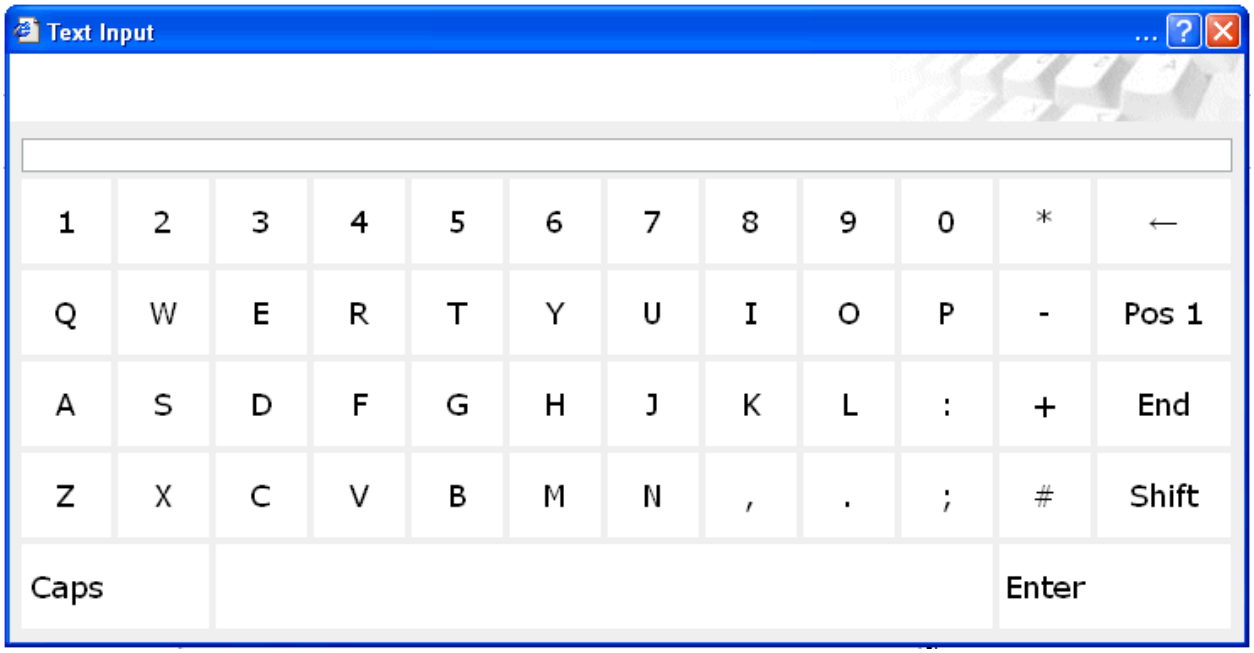
Consequence: inside your reaction on the value help request (e.g. in your implementation of `findValidValuesFor...()`), you can already filter the valid values against what the user has already entered.

Touch Screen Support

As mentioned in the property list, the field is able to offer touch screen support. Have a look at the following example:

If the user clicks into the area in which you can see a keyboard shining through as background, the user will get one of the following pop-ups - depending on the data type assigned to the FIELD control.


Number Input
7
8
9
4
5
6
1
2
3
0
,
Del
Enter



The XML layout definition is:

```
<rowarea name="Demo">
    <itr>
        <label name="Integer Input" width="100">
        </label>
       <field valueprop="intValue" width="150" touchpadinput="true" datatype="int">
        </field>
    </itr>
    <itr>
        <label name="Float Input" width="100">
        </label>
        <field valueprop="floatValue" width="150" touchpadinput="true" ↩
datatype="float">
        </field>
    </itr>
    <itr>
        <label name="Text Input" width="100">
        </label>
        <field valueprop="stringValue" width="250" touchpadinput="true">
        </field>
    </itr>
</rowarea>
```

In all FIELD controls, the property `touchpadinput` is set to "true". The server side adapter processing does not differ in any way from the normal adapter processing.

Properties

Basic			
valueprop	Server side property representation of the control.	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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
length	Width of FIELD in amount of characters. WIDTH and LENGTH should not be used together. Note that the actual size of the control depends on the font definition if using the LENGTH property.	Optional	5 10 15 20 int-value
maxlength	Maximum number of characters that a user may enter. This property is not depending on the LENGTH property - please do not get confused by the similar naming. MAXLENGTH has	Optional	5 10 15

	nothing to do with the optical sizing of the control but only with the number of characters you may input.		20 int-value
autotab	If set to true, an automatic tab is executed for fields with a specified MAXLENGTH when the maxlength value is reached. For fields without a MAXLENGTH specified it has no effect. Default is true.	Optional	true false
textalign	Alignment of text inside the control.	Optional	left center right
password	If set to "true", each entered character is displayed as a '*'.	Optional	true false
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
direction	Presets the default(BiDi) direction of the control. Use black string in order to have the default value.	Optional	rtl ltr
uppercase	If "true" then all input is automatically transferred to upper case characters.	Optional	true false
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column.	Optional	top

	Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.		middle bottom
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
fieldstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	mouse-button in your browser and select the "View source" or "View frame's source" function.		
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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2
noborder	Boolean value defining if the control has a border. Default is "false".	Optional	true false
transparentbackground	Boolean value defining if the control is rendered with a transparent background. Default is "false".	Optional	true false
bgcolorprop	Property of the adapter object to provide the background color of the control.	Optional	
fgcolorprop	Name of adapter property that passes back a color value (e.g. "#FF0000" for red color). The color value is used as text color in the control. - The background color is automatically chosen dependent from the text color: for light text colors the background color is black, for dark text colors the color is default. Use BGCOLORPROP to choose both - text and background color.	Optional	
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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
Binding			
valueprop	(already explained above)		
alwaysflush	If set to TRUE then a specified server flushmethod is also called in case the value has not changed. The default is FALSE, meaning that a server flushmethod is only called for a changed value.	Optional	true false
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
flushindexprop	$en/popupwizard/_attr_flushindexprop$	Optional	

contextmenu	If set to TRUE for a field myfield, method/event reactOnContextMenuMyfield will be called/triggered on right mouse click. In this method/event you can set a contextmenu correspondingly. Please use the attribute CONTEXTMENUMETHOD in case you would like to use a different method/eventname. In case a valid value is specified for the CONTEXTMENUMETHOD attribute, the value for the CONTEXTMENU attribute is ignored. Default value is FALSE.	Optional	true false
contextmenumethod	Name of the method on adapter level that is called when the user presses the right mouse button in an empty area.	Optional	
onclickmethod	Name of the method inside the row item class that is called if the user clicks.	Optional	
ondblclickmethod	Name of the method that is called when the user double clicks.	Optional	
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
valuetextprop	Name of the adapter property that provides a "human understandable" description for the value: in some cases you enter an id into a FIELD but want to display the id and a description to the user. At runtime, the values provided by the VALUEPROP- and the VALUETEXTPROP-property are combined into one text (string) that is returned into the FIELD.	Optional	
textidmode	If using property "valuetextprop" then a field knows an id and a text for a certain value. There are three types of display: either both are shown together, separated by an "-" (e.g. "id - text"). Or only text is shown or only the id is shown. If not defined at all then the system's default text id-mode will be chosen. The default mode can be defined as part of the CIS session context.	Optional	0 1 2
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	

bgcolorprop	(already explained above)		
fgcolorprop	(already explained above)		
autocallpopupmethod	Use property AUTOCALLPOPUPMETHOD to invoke the field's value help method with a certain offset (milliseconds) after last key down event	Optional	true false
lengthprop	Property of the adapter that defines the width of FIELD in amount of characters.	Optional	
maxlengthprop	Name of adapter property that passes back the maximum number of characters that a user may enter. Consider to use MAXLENGTH to define this number in a static way.	Optional	
Validation			
datatype	By default, the FIELD control is managing its content as string. By explicitly setting a datatype you can define that the control... ...will check the user input if it reflects the datatype. E.g. if the user inputs "abc" into a field with datatype "int" then a corresponding error message will popup when the user leaves the field. ...will format the data coming from the server or coming form the user input: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). In addition value popups are offered for the user automatically for some datatypes: e.g. when specifying datatype "date" the automatically the field provides a calendar input popup. Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	date float int long time timestamp color xs:decimal xs:double xs:date xs:dateTime xs:time ---------- N n.n P n.n string n L xs:boolean xs:byte

			xs:short
editmask	NATPAGE only: A subset of the Natural edit masks is supported depending on the data type.	Optional	
validationrules	Contains information used for Data Validation. Use the Validation Rules Editor to make changes!	Optional	
validation	Regular expression against which the content of the field is checked on client side when the user changes the field. If the validation fails then an error message popup up and informs the user about the wrong input.	Optional	[a-zA-Z0-9_.-] {1,}\ \@[a-zA-Z0-9_.-] {1,}\ \.\ \w{2,}\ \d{5} [0-9)(-/+]+
validationprop	Property out of which the regular expression is dynamically read. Works the same way as VALIDATION but in a dynamic way.	Optional	
validationuserhint	If a client side validation fails due to wrong user input then an error popup is opened. If you define a hint inside this property then the hint is output to the user in order to tell in which way to input the value. The hint is not language dependent.	Optional	
validationuserhintprop	If using validation expressions (either property "validation" or "validationprop") then a popup comes up if the user inputs wrong values into a field. Inside this popup a certain text may be added in order to explain to the user what he/she did not correctly input. This text can be either statically defined or dynamically - by using this property reference.	Optional	
digits	Number that specifiies how many digits are to be displayed (i.e. digits before the comma). If using this feature then the DATATYPE property must be set to 'float'. See also DECIMALDIGITS.	Optional	1 2 3 int-value
digitsprop	Property of the adapter that passes back information how many digits are to be displayed (i.e. digits before the comma). If using this feature then the DATATYPE property must be set to 'float'.	Optional	

decimaldigits	Specifies the number of displayed decimal digits. If using this feature then the DATATYPE property must be set to 'float'.	Optional	1 2 3 int-value
decimaldigitsprop	Property of the adapter that passes back information how many decimal digits are to be displayed. If using this feature then the DATATYPE property must be set to 'float'.	Optional	
spinrangemin	An integer value which defines the lower bound of the value range.	Optional	1 2 3 int-value
spinrangemax	An integer value which defines the upper bound of the value range.	Optional	1 2 3 int-value
Valuehelp			
popupmethod	Name of the adapter's method that is called when the user requests value help by pressing F4 or F7 or by clicking into the FIELD with the right mouse button. See at chapter 'Popup Dialog Management' for more details. If the POPUPMETHOD is defined, a small icon is shown inside the field to indicate to the user that there is some value help available.	Optional	openIdValueCombo openIdValueHelp openIdValueComboOrPopu
popupinputonly	Boolean property that control if a field with POPUPMETHOD defined is still usable for keyboard input. If "false" (= default) then the user can input a value either directly via keyboard or by using the popupmethod's help. If set to "true" then no keyboard input is possible - but only selection from the popup-method's help.	Optional	true false
popupprop	Name of an adapter's boolean property to provide information whether a POPUPMETHOD is available (true) or not (false). This feature is used in scenarios in which	Optional	

	a FIELD offers e.g. value help or not, depending on business logic inside the adapter.		
popuponalt40	Value help in a field is triggered either by clicking with the mouse or by pressing a certain key inside the field. The "traditional" keys are "cursor-down", "F7" or "F4". Sometimes you do not want to mix other "cursor-down" behaviour (e.g. scrolling in lists) with the value help behaviour. In this case switch this property to "true" - and the value help will only come up anymore when "alt-cursor-down" is pressed.	Optional	true false
popupcombowidth	Pixel width of the standard "openIdValueCombo" popup dialog. Default is field width or at least 150 pixel.	Optional	1 2 3 int-value
popupicon	URL of image that is displayed inside the right corner of the field to indicate to the user that there is some value help available.. 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. specifying "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	gif jpg jpeg
touchpadinput	Boolean property that decides if touch pad support is offered for the FIELD control. The default is "false". If switched to "true" then you can input data into the field via a touch pad. As consequence you can use this control for making inputs through a touch terminal.	Optional	true false
onlinehelp			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
title	Text that is shown as tooltip for the control.	Optional	

	Either specify the text "hard" by using this TITLE property - or use the TITLETEXTID in order to define a language dependent literal.		
titletextid	Text ID that is passed to the multi language management - representing the tooltip text that is used for the control.	Optional	
placeholder	The text for the HTML placeholder attribute. The placeholder attribute specifies a short hint that describes the expected value.	Optional	
formula	Contains information used by the Formula Editor. Use the Formula Editor to make changes!	Optional	
Hot Keys			
hotkeys	Semicolon separated list of hot keys. A hotkey consists of a list of keys and a method name. Separate the keys by "-" and the method name again with a semicolon Example: ctrl-alt-65;onCtrlAltA;13;onEnter ...defines two hot keys. Method onCtrlAltA is invoked if the user presses Ctrl-Alt-A. Method "onEnter" is called if the user presses the ENTER key. Use the popup help within the Layout Painter to input hot keys.	Optional	
Add-ons			
autocompleteref	Adds autocomplete functionality to the FIELD control. As value set the id of the AUTOCOMPLETE control.	Optional	
autocompletedropdownicon	Set this property to use your own dropdown icon. URL of an icon, which is used to show the whole list when it is clicked.	Optional	gif jpg jpeg
autocompletedisplayname	Name of the value to be displayed in an additional control.	Optional	
autocompletedisplayref	Sets a reference to an additional control to display additional information on selection. As value set the valueprop of the control in which you would like to display the information.	Optional	
autocompleteresultsref	Sets a reference to an additional control to display the total number of results. Use this when the number of matching items can be very	Optional	

	high and you limited the number of displayed items in the dropdown for performance reasons. As value set the valueprop of the control in which you would like to display the total number.		
autocompletewithdropdown	If set to "TRUE" a dropdown button/icon will be appended to the field. When it is clicked, all items are shown.	Optional	true false
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	
autocallpopupmethodoffset	The offset (milliseconds) after the last key down event for calling the AUTOCALLPOPUPMETHOD. Makes only sense if an AUTOCALLPOPUPMETHOD is specified.	Optional	1 2 3 int-value
formautocomplete	This property only has effects if the withformtag property in the PAGEBODY is activated. In this case you can switch on and off the browser's autocomplete behavior for HTML form tags in single FIELD controls. Default is TRUE.	Optional	on off new-password

32 FILEUPLOAD/FILEUPLOAD2

The file upload controls simplify the process of uploading files from the client to the server. Two types are available:

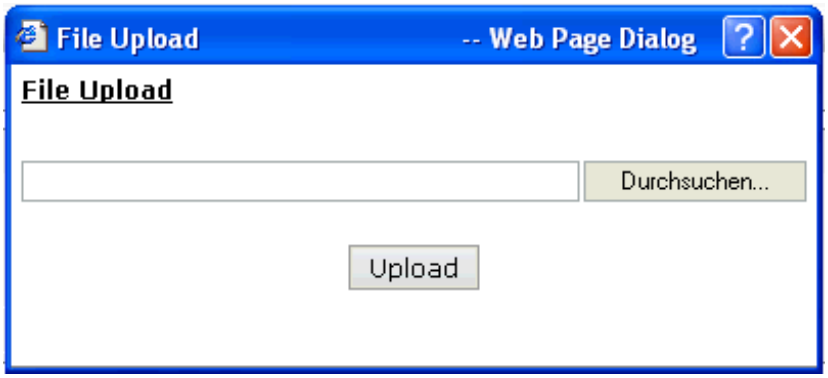
- The FILEUPLOAD control is represented by a button. When you choose the button, a dialog appears showing the file upload form (field input and a file selection button).
- With the FILEUPLOAD2 control, you embed the file upload form into your page.

Both types have the program binding, i.e. you can switch between the two types without touching your code.

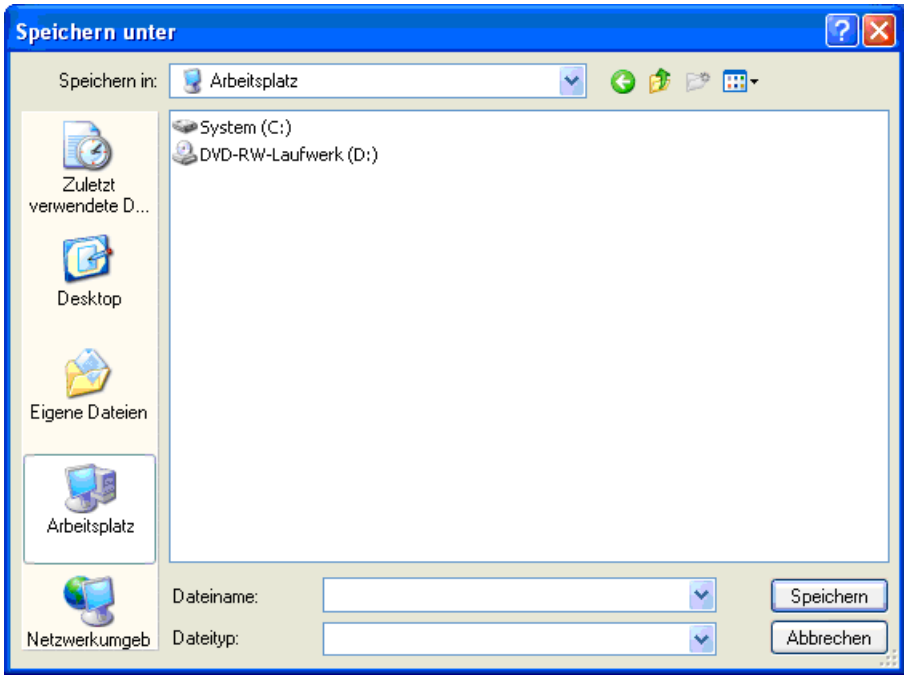
FILEUPLOAD

The FILEUPLOAD control simplifies the process of uploading files from the client to the server. Look at the following example:

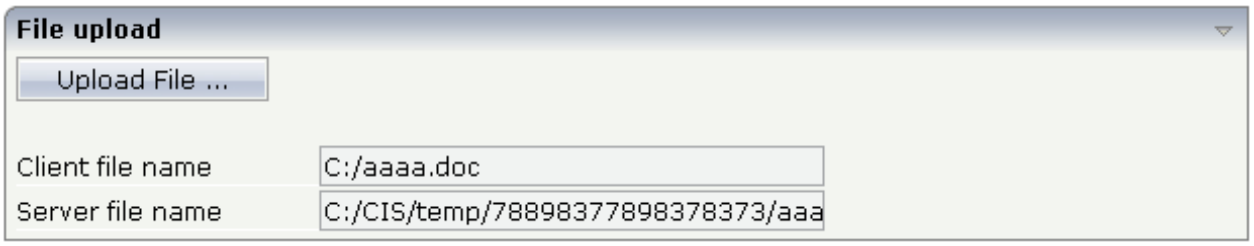
The control - from the look-and-feel perspective - is a button with some special reaction. When you choose the button, the following dialog appears:



You can either enter a file name or you can invoke the file selection dialog by choosing the button to the right of the field (which appears in the language of the browser).



After choosing the **Upload** button, the first screen looks as follows:



Have a look at the layout definition:

```
<rowarea name="File upload">
    <itr>
        <fileupload name="Upload File ..." cfileprop="clientFileName"
                    sfileprop="serverFileName" method="onUploadFile">
        </fileupload>
    </itr>
    <vdist height="20">
    </vdist>
    <itr>
        <label name="Client file name" width="150">
        </label>
        <field valueprop="clientFileName" width="250" displayonly="true">
        </field>
    </itr>
    <itr>
        <label name="Server file name" width="150">
        </label>
```

```
        <field valueprop="serverFileName" width="250" displayonly="true">
        </field>
    </itr>
</rowarea>
```

The FILEUPLOAD control references to two properties. The `cfileprop` property references to a property in which the client file name of the uploaded file is written. The `sfileprop` property references to a property in which the server file name of the uploaded file is written; after the upload is finished, the file is copied into a server-side directory of your Application Designer installation and is accessible by normal file I/O operations.

With the `method` property, you define which method is called when a file is uploaded.

Each file gets a unique file name on the server - you have full access to the file and you can process the file as you need it, e.g. you can also delete it after usage.

The Java code looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class FileUploadAdapter
    extends Adapter
{
    // property >clientFileName<
    String m_clientFileName;
    public String getClientFileName() { return m_clientFileName; }
    public void setClientFileName(String value) { m_clientFileName = value; }

    // property >serverFileName<
    String m_serverFileName;
    public String getServerFileName() { return m_serverFileName; }
    public void setServerFileName(String value) { m_serverFileName = value; }

    /** */
    public void onUploadFile()
    {
        outputMessage(MT_SUCCESS, "Upload was successful!");
    }
}
```

Usage of `com.softwareag.cis.file.FileManager`

See the Java API documentation for `com.softwareag.cis.file.FileManager`. If you do not have file access routines on your own, you may use this one in order to read the file created on the server when using the upload button. For example, it offers the methods:

- `byte[] readFileIntoByteArray`
- `String[] readTextFileIntoStrings`

- `String readTextFileIntoString/readTextFileIntoStringWithLineBreak`

Location of Server File

The result of the upload processing is the generation of a server side file. Why do we pass the information as a file and not as an in-memory byte array? Reason: by doing so, we do not get problems if your application server runs out of memory when uploading very large files. Only a part of the upload file is in memory, i.e. when uploading, the server file is written in certain blocks - the memory blocked by the upload processing has a maximum size of this block (16 KBytes).

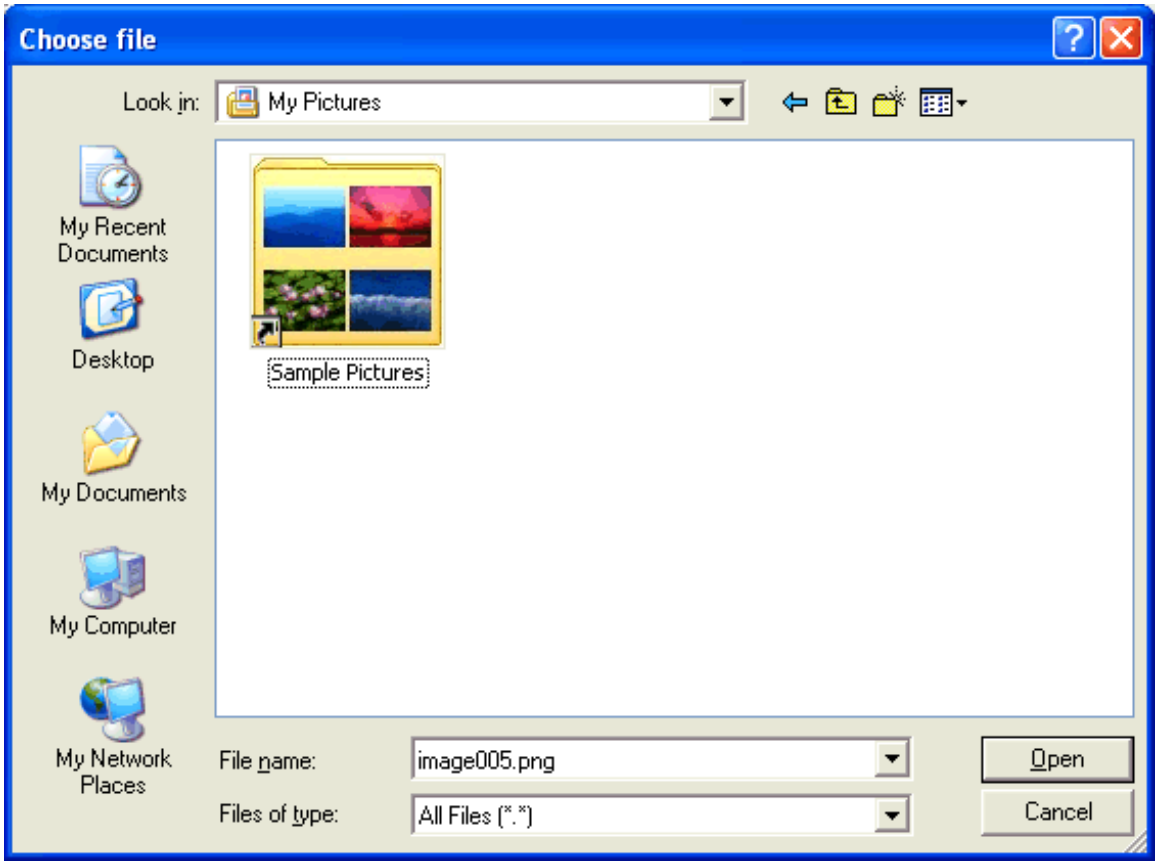
The location of the file is the temporary directory that is provided by the servlet container in which the web application that uses Application Designer runs.

Note: Part of the Servlet 2.2 protocol is the definition of the ServletContext. Part of this API is the obligatory API that passes a temporary directory to web applications.

FILEUPLOAD2

With the FILEUPLOAD2 control, you embed the file upload form into your page.

You can either enter a file name or you can invoke the file selection dialog by choosing the button to the right of the field (which appears in the language of the browser).



After choosing the file, the screen looks as follows:

Have a look at the layout definition:

```
<rowarea name="Upload...">
   <itr>
        <fileupload2 width="300" cfileprop=" clientFileName" ↩
sfileprop="serverFileName" method=" onUploadFile">
       </fileupload2>
    </itr>
</rowarea>
```

The FILEUPLOAD2 control references to two properties. The `cfileprop` property references to a property in which the client file name of the uploaded file is written. The `sfileprop` property references to a property in which the server file name of the uploaded file is written; after the upload is finished, the file is copied into a server-side directory of your Application Designer installation and is accessible by normal file I/O operations.

With the `method` property, you define which method is called when a file is uploaded.

Each file gets a unique file name on the server - you have full access to the file and you can process the file as you need it, e.g. you can also delete it after usage.

The Java code looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class FileUploadAdapter
    extends Adapter
{
    // property >clientFileName<
    String m_clientFileName;
    public String getClientFileName() { return m_clientFileName; }
    public void setClientFileName(String value) { m_clientFileName = value; }

    // property >serverFileName<
    String m_serverFileName;
    public String getServerFileName() { return m_serverFileName; }
    public void setServerFileName(String value) { m_serverFileName = value; }

    /** */
    public void onUploadFile()
    {
        outputMessage(MT_SUCCESS, "Upload was successful!");
    }
}
```

Note: The coding is exactly the same when using the FILEUPLOAD control. You can switch between the two types (FILEUPLOAD and FILEUPLOAD2) without touching your code.

FILEUPLOAD 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	
cfileprop	Name of adapter property in which the client's file name is passed at upload time.	Obligatory	

sfileprop	Name of adapter property, in which at upload time - the name of the file is written - which is a copy of the client file but in the server's file system. Please pay attention that this file name is a unique one and has nothing to do with the client's file name!	Obligatory	
method	Method called inside the adapter when a file is uploaded. The file's data is available at the point of time this method is called.	Obligatory	
maxfilesize	Maximum size of the file. The unit is specified in the filesizeunit property.	Optional	1 2 3 int-value
filesizeunit	The unit for the maxfilesize property. Default is bytes.	Optional	bytes KB MB GB
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
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. specifying "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	gif jpg jpeg
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 percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the	Optional	100 120 140 160 180 200 50%

	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.		100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
invisiblemode	This property has three possible values: (1) "invisible": the control is not visible without occupying any space. (2) "disabled": the control is deactivated: it is "grayed" and does not show any roll over effects any more. (3)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
buttonstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press	Optional	

	right mouse-button in your browser and select the "View source" or "View frame's source" function.		
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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			
cfileprop	(already explained above)		

sfileprop	(already explained above)		
method	(already explained above)		
visibleprop	(already explained above)		
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

FILEUPLOAD2 Properties

Basic			
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 percentage 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%
cfileprop	Name of adapter property in which the client's file name is passed at upload time.	Optional	
sfileprop	Name of adapter property, in which at upload time - the name of the file is written - which is a copy of the client file but in the server's file system. Please pay attention that this file name is a unique one and has nothing to do with the client's file name!	Optional	
method	Method called inside the adapter when a file is uploaded. The file's data is available at the point of time this method is called.	Optional	
maxfilesize	Maximum size of the file. The unit is specified in the filesizeunit property.	Optional	1 2 3

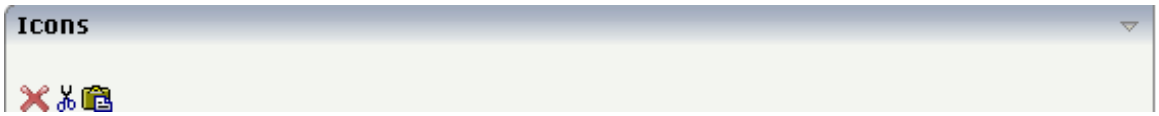
			int-value
filesizeunit	The unit for the maxfilesize property. Default is bytes.	Optional	bytes KB MB GB
withsubmitbutton	If set to "TRUE" adds an additional button to the control to start the file upload.	Optional	true false
submitbuttonname	The name of the submit button in case WITSUBMITBUTTON is set to "true".	Optional	
submitbuttontextid	"Textid" for the name of the submitbutton if WITHSUBMITBUTTON is set to "true". The "textid" is translated into a corresponding string at runtime. Do not specify WITHSUBMITBUTTONNAME if specifying WITHSUBMITBUTTONID.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
cfileprop	(already explained above)		
sfileprop	(already explained above)		
method	(already explained above)		
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible disabled cleared
Appearance			
invisiblemode	(already explained above)		
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	Optional	1 2 3

	you may want to define the control to span over more than one columns. The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		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 synchronized 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
darkbackground	Normally the background is in light colour but the CIS style sheets also have a dark(er) grey colour to be used. If DARKBACKGROUND is set to true then the darker background colour is chosen. This property typically is used to integrate light coloured controls into darker container areas.	Optional	true false
Online Help			
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	

33 ICON

The ICON control is similar to the BUTTON control, but it uses an image to display its function. When chosen, it calls a method in the adapter class.

Example



The XML layout definition is:

```
<rowarea name="Icons">
    <itr>
        <icon image="../HTMLBasedGUI/images/remove.gif" method="remove" ↩
title="Remove">
        </icon>
       <icon image="../HTMLBasedGUI/images/cut.gif" method="cut" withdistance="true"
              title="Cut">
        </icon>
       <icon image="../HTMLBasedGUI/images/paste.gif" method="paste" title="Paste">
        </icon>
    </itr>
</rowarea>
```

Hiding and Disabling Icons

As with many other controls, the icon provides an `invisibleprop` property that may point to an adapter property that decides whether to display an icon ("true") or not ("false"). By using the property `imageinactive`, you can fine-control the icon's behavior in the following way:

- When defining an image in `imageinactive`, then this image will replace the icon's image that is normally displayed. The icon itself will be inactive, i.e. there are no roll-over effects and there is no possibility to click on it.
- When not defining an image in `imageinactive`, then the icon will be hidden.

Consequence: if you want to show grayed images for inactive icons, then use `imageinactive`.

Properties

Basic			
image	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Obligatory	gif jpg jpeg
imagertl	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Optional	gif jpg jpeg
method	Method of your adapter object that is executed when clicking on the control.	Obligatory	
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.	Optional	
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".	Optional	

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
imagewidth	Pixel width of the image that is shown inside the icon. If not defined then the icon is rendered with its normal width.	Optional	10 20 40 100 300
imageheight	Pixel height of the image that is shown inside the icon. If not defined then the icon is rendered with its normal height.	Optional	10 20 40 100 300
textsize	The HTML font size of the text. Corresponding to the HTML definition "1" means "smallest" and "6" means "biggest".	Optional	1 2 3 4 5 6
imageinactive	If the visibility is dynamically controlled by using the INVISIBLEPROP then there are two ways the icon reacts if the corresponding property passes back "false". If you want the icon to switch into an inactive status then define inside this property the URL of the image that is the inactive counter part to the normal icon image. Maybe the image is a grayed version of the normal icon image. If you do not define a value for this property then the icon is made invisible.	Optional	gif jpg jpeg

align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
colstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
spanstyle	CSS style definition that is directly passed into this control.	Optional	background-color: #FF0000

	With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		color: #0000FF font-weight: bold
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
switchvisibleproponuserinput	If set to TRUE, the visibleprop is automatically switched to TRUE in case of user input to any input control in this page. The 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
nameposition	Position of the (optional) text to the icon. Aside or below, default is aside. Set the corresponding text in the name or the text id property.	Optional	aside below

displaymenuindicator	If set to true a small indicator signals that there is a corresponding menu 'behind this icon'. Default is false.	Optional	true false
Binding			
method	(already explained above)		
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
imageprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
imageinactiveprop	Name of adapter property that provides as value the URL of the image that is shown when the control is inactive.	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	
titleprop	(already explained above)		
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	
Deprecated			
withdistance	If set to "true" then 2 pixels of distance are kept on the left and on the right of the icon.	Optional	true false

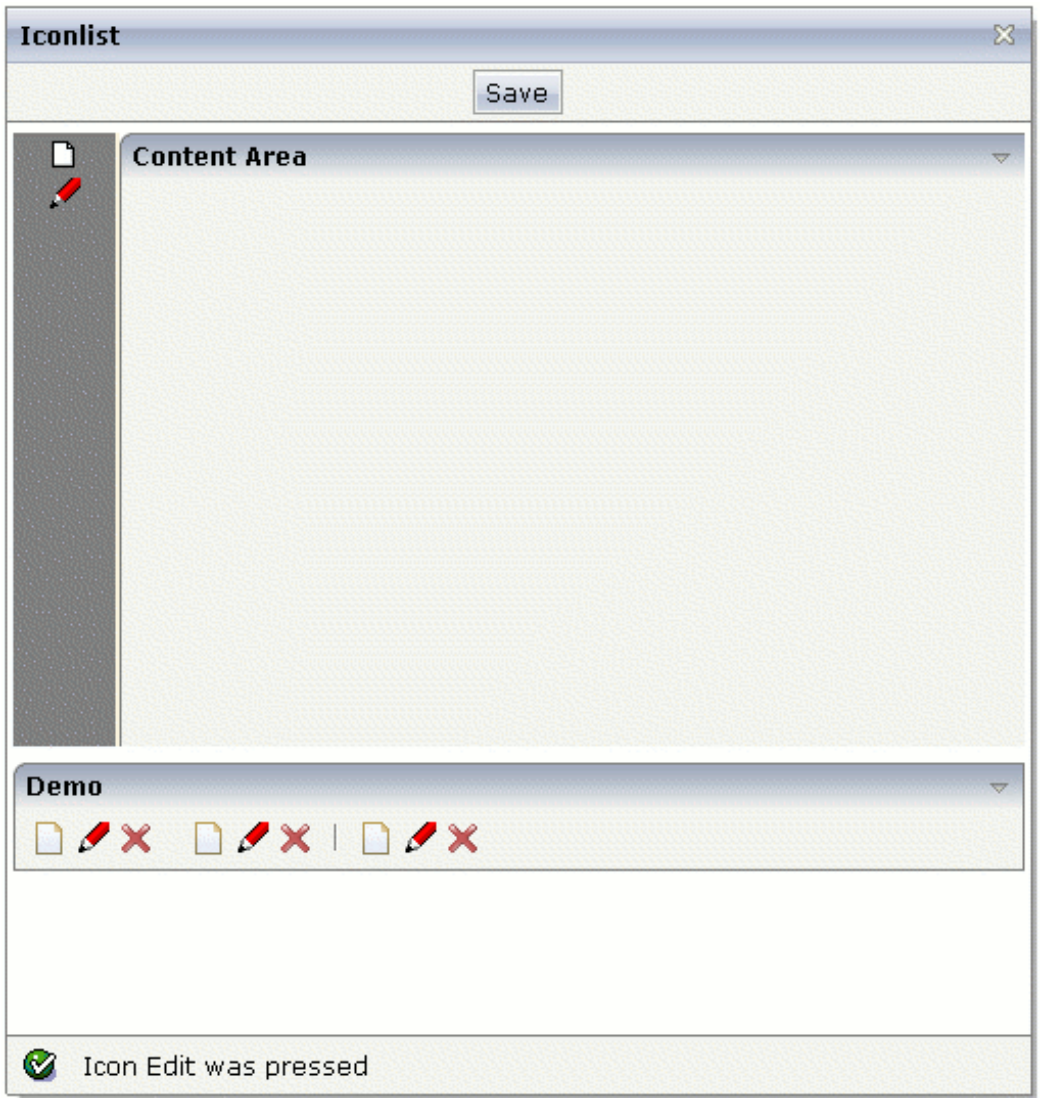
	Reason behind: if arranging several icons inside one table row (ITR, TR) then a certain distance is kept between the icons when this property is set to "true".		

34 ICONLIST

The ICONLIST is very similar to the BUTTONLIST, representing a list of items instead of a list of buttons. The list can either be a vertical list or a horizontal list.

Example: Vertical Icon List

This example is already a complex example. Additional style information was added to make the icon list look like a bar which provides icons for favorites.



The XML layout definition is:

```
<page model="Icon_ListAdapter">
    <titlebar name="Iconlist">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
        <vdist height="5">
        </vdist>
        <itr takefullwidth="true" height="300">
            <coltable0 width="50" takefullheight="true" fixlayout="true">
                <iconlist iconlistprop="iconList" cellspacing="3"
                          tablestyle="background-color:#808080">
                </iconlist>
                <vdist height="100%" backgroundstyle="background-color:#808080">
                </vdist>
            </coltable0>
            <coltable0 width="2" takefullheight="true" fixlayout="true"
                       tablestyle="background-color:#F7F3DE">
            </coltable0>
            <coltable0 width="100%" takefullheight="true" fixlayout="true">
                <rowarea name="Content Area" height="100%" withrightborder="false"
                         withbottomborder="false" withtoppadding="false"
                         areastyle="border-top: 0px">
                </rowarea>
            </coltable0>
        </itr>
        <vdist height="5">
        </vdist>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The server side adapter code that dynamically builds up the list of icons is:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.ICONLISTInfo;
import com.softwareag.cis.server.util.ICONLISTItem;

// This class is a generated one.

public class Icon_ListAdapter
    extends Adapter
{
    // class >MyICONLISTItem<
    public class MyICONLISTItem extends ICONLISTItem
    {
        public MyICONLISTItem(ICONLISTInfo info, String imageURL, String text)
        {
            super(info, imageURL, text);
```

```
        }

        public void execute()
        {
            outputMessage(MT_SUCCESS,"Icon " + getText() + " was pressed");
        }
    }

    // property >iconList<
    ICONLISTInfo m_iconList = new ICONLISTInfo();
    public ICONLISTInfo getIconList() { return m_iconList; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        // Fill IconList
        MyICONLISTItem item;
        item = new MyICONLISTItem(m_iconList,"images/new.gif","New");
        item = new MyICONLISTItem(m_iconList,"images/edit.gif","Edit");

    }
}
```

Similar to the BUTTONLIST control, each icon is represented by a single object. All objects are collected in an `ICONLISTInfo` object.

Example: Horizontal Icon List

By setting the property `vertical` to "false", you can build horizontal icon lists.

The layout definition is:

```
<rowarea name="Demo">
    <iconlist iconlistprop="iconList_02" vertical="false" cellspacing="3">
    </iconlist>
</rowarea>
```

The code for creating the icon list is:

```
public void init()
{
    // Fill IconList 02
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/new.gif","New");
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/edit.gif","Edit");
    item = new ↩
MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/remove.gif","Remove");
    m_iconList_02.addDistance();
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/new.gif","New");
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/edit.gif","Edit");
    item = new ↩
MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/remove.gif","Remove");
    m_iconList_02.addSeparator();
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/new.gif","New");
    item = new MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/edit.gif","Edit");
    item = new ↩
MyICONLISTItem(m_iconList_02,"../HTMLBasedGUI/images/remove.gif","Remove");
}
```

Properties

Basic			
iconlistprop	Name of adapter property representing the control on server side. The property must be of type ICONLISTInfo. Read further information inside the Java API Documentation.	Obligatory	
vertical	Direction of the icon list. If not specified (or set to "true") then the icons are arranged in one column, one below the other. If specified as "false" then the icons are arrange in one row, one aside the other.	Optional	true false
cellspacing	An icons of the ICONLIST control is embedded into an internal cell. The CELLSPACING property defined the number of pixels that are kept between the icon an the border of this cell. Use the CELLSPACING in order to define a certain distance each icon keeps from the next item.	Optional	1 2 3 int-value
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			

imagewidth	Pixel width of the image that is shown inside the icon. If not defined then the icon is rendered with its normal width.	Optional	10 20 40 100 300
imageheight	Pixel height of the image that is shown inside the icon. If not defined then the icon is rendered with its normal height.	Optional	10 20 40 100 300
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
tablestyle	Style definition (following CSS style sheet definitions) that is used for the background area of the ICONLIST control.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
cellstyle	Style definition (following CSS style sheet definitions) that is used for each cell area of the ICONLIST control in which an icon is kept.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

displaymenuindicator	If set to true a small indicator signals that there is a corresponding menu 'behind this icon'. Default is false.	Optional	true false
additionaltextposition	Position of the text that is displayed inside the control. Use method ICONLISTItem.setName to set the text.	Optional	aside below
textsize	The HTML font size of the text. Corresponding to the HTML definition "1" means "smallest" and "6" means "biggest".	Optional	1 2 3 4 5 6
withrightpadding	Flag (boolean) that indicates whether to insert a padding right hand of the last icon. This attribute does apply for horizontal ICONLIST only (see attribute VERTICAL). Default is true.	Optional	true false
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	

35 IHTML

The IHTML control is used to embed server side generated HTML inside a page that is provided by an adapter property. The IHTML control is very flexible on the one hand. On the other hand, you have to take care about what is defined inside the IHTML area.

Use this control if you have, for example, a server side report generation program already producing HTML as output which you want to include into your pages, etc.

Example

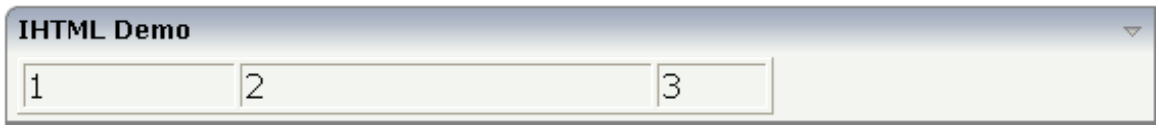
Layout definition:

```
<rowarea name="IHTML Demo">
    <itr>
        <ihtml valueprop="generatedHTML">
        </ihtml>
    </itr>
</rowarea>
```

Program code:

```
// property >generatedHTML<
String m_generatedHTML;
public String getGeneratedHTML()
{
    return "<table border=1>"+
           "<tr>"+
           "<td width=100>1</td>"+
           "<td width=200>2</td>"+
           "<td width=50>3</td>"+
           "</tr>"+
           "</table>";
}

public void setGeneratedHTML(String value) { m_generatedHTML = value; }
```

The above layout definition, together with the above program code, produces the following page:



Note that the IHTML control internally opens a table cell in which you place the HTML code coming from the adapter property. Your HTML must fit into this concept, i.e. it must be embeddable into an HTML arrangement like the following:

```
...
...
    <table ...>
    ...
    <tr>
        ...
        <td>
            <!-- THIS IS WHERE YOUR HTML NEEDS TO FIT IN -->
        </td>
        ...
    </tr>
    ...
    </table>
...
```

Pros and Contras when Using the IHTML Control

The IHTML control is powerful because it offers all possibilities to directly embed HTML code inside your page. The “pros” are:

- You can render complex HTML statements on the server side. The client just plugs the HTML into the page. The rendering effort on the client side is very low.
- You can directly write HTML on the server side. If you are an HTML expert, then you may like to do so in certain situations.

On the other hand, there are “cons”:

- You begin to write rendering logic on your own - on the server side.
- You have to take care of being compatible through various browsers.
- You have to take care of a consistent appearance of the HTML results in order to have a consistent look and feel throughout your application.
- You may increase the traffic over the network.

Conclusion:

- HTML rendering on the server side should not be offered in general, but only under defined circumstances.
- Encapsulate your server side rendering behind a proper Java interface. Application developers work with the Java interface; the HTML rendering behind should be done by an HTML expert.

Application Designer also uses the IHTML control internally: in the area of reporting, Application Designer provides a server side Java API for simple report generation. Behind the report interface, HTML (or PDF) is generated which is passed into an internally used IHTML control.

Scripting in Generated HTML

Based on the conclusion of the previous section, you should be aware of the fact that it is also possible to add JavaScript statements into the HTML code that you define for the IHTML control. See *Custom Controls* for information on the JavaScript API of Application Designer. You can call the JavaScript library functions from the script that you pass as part of the HTML you create.

Example: you may invoke a method behind a certain link. Your program might look like this:

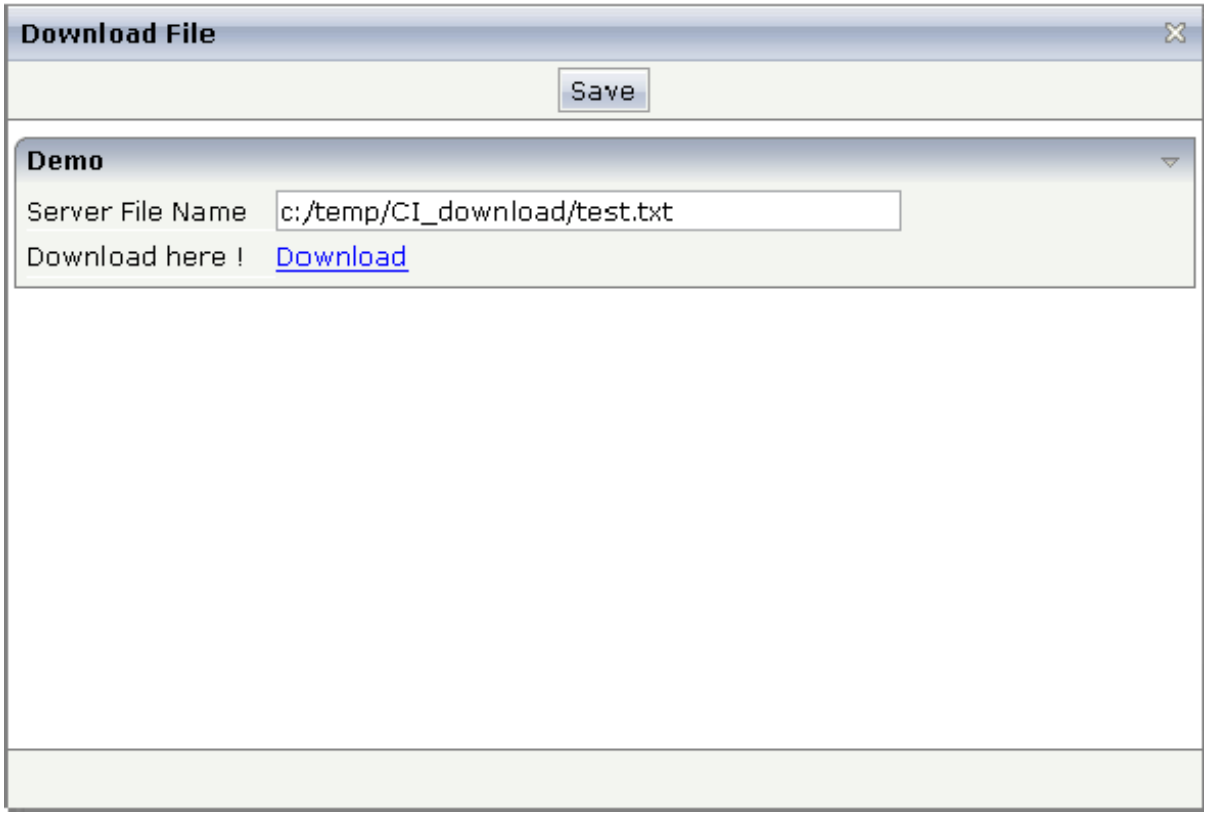
```
String m_html;

public String getHtml() { return m_html; }

public void createHTMLString()
{
    ...
    m_html += "<a href='javascript: ↩
csciframe.invokeMethodInModel(\'onClick\');'>Hello</a>";
    ...
}
```

Example: Building Download Links

While HTML provides an explicit file upload feature (which is internally used when using Application Designer's UPLOAD control), there is no explicit download control to load server data onto the client machine. However, there is the standard behavior of the browser when following a link.

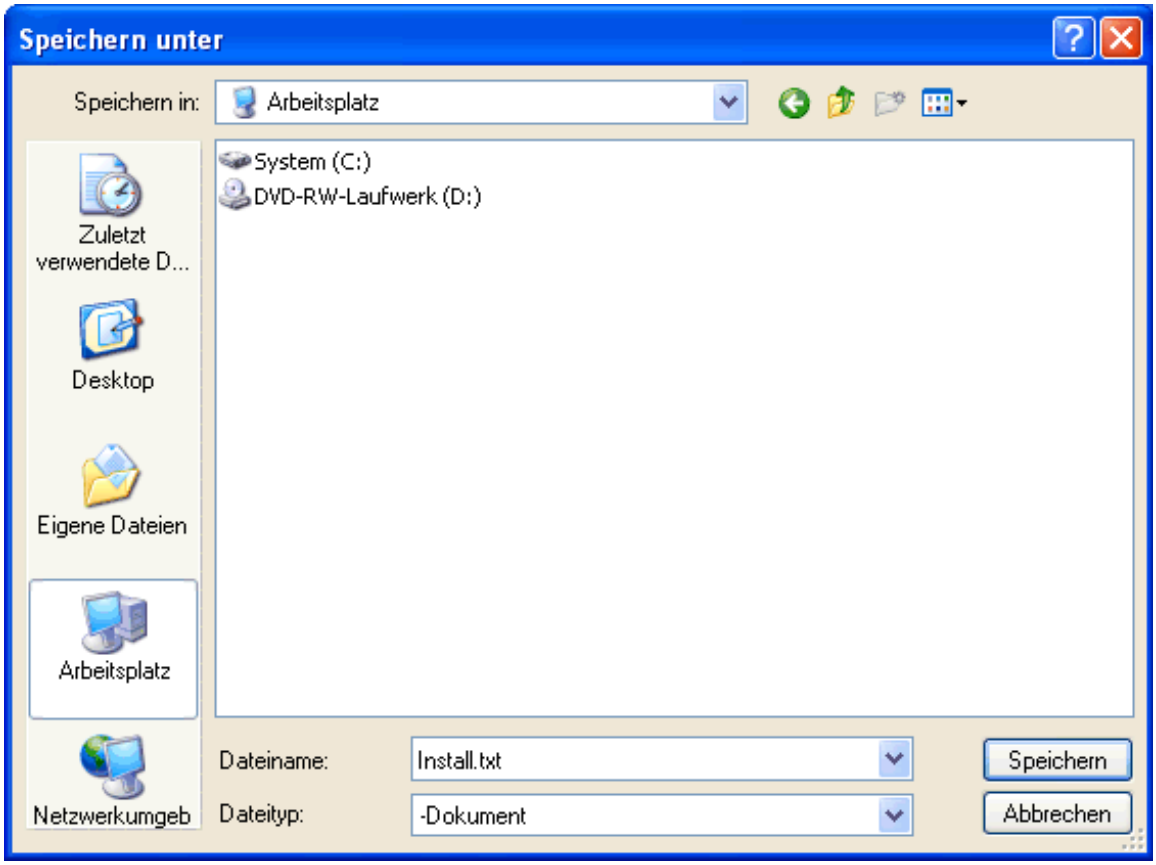
The browser knows certain document content types (MIME types). For example, the browser knows that PDF documents are displayed in an Adobe Reader plug-in. If the browser receives a URL with a content type that the browser does not understand, then it automatically opens a pop-up in which the user is asked to download the file.

Have a look at the following screen:



The user enter the name of a file that resides on the server (in this case, it is a Windows-based server). After entering the name, the user sees a **Download** link. When choosing this link, the following pop-up appears.

In the pop-up, the user is asked to save the file in the local file system of the browser. After this security message, the typical file selection dialog box appears:



The XML page layout looks as follows:

```
<page model="DownloadAdapter">
    <titlebar name="Download File">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
        <vdist height="5">
        </vdist>
        <rowarea name="Demo">
            <itr>
                <label name="Server File Name" width="120">
                </label>
                <field valueprop="serverfilename" width="300" flush="server">
                </field>
            </itr>
            <itr>
                <label name="Download here !" width="120">
                </label>
                <ihtml valueprop="downloadURL">
                </ihtml>
```

```
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You see that within the FIELD, the property `serverFileName` is maintained. The FIELD directly flushes value changes to the server. The HTML text that is shown inside the IHTML control, is provided by the property `downloadURL`.

The adapter code is:

```
import com.softwareag.cis.file.FileManager;
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class DownloadAdapter
    extends Adapter
{
    // property >downloadURL<
    String m_downloadURL;
    public String getDownloadURL() { return m_downloadURL; }
    public void setDownloadURL(String value) { m_downloadURL = value; }

    // property >serverfilename<
    String m_serverfilename="c:/temp/CI_download/test.txt";
    public String getServerfilename() { return m_serverfilename; }
    public void setServerfilename(String value)
    {
        m_serverfilename = value;
    // load file
        try
        {
        // SECURITY!!! Only allow download from one certain directory
        m_serverfilename = m_serverfilename.replace('\\','/');
        if (!m_serverfilename.startsWith("c:/temp/CI_download/") &&
        !m_serverfilename.startsWith("/tmp/CI_download/"))
        {
        throw new Exception("Only can download files from dedicated directories!");
        }
        // read file content
        byte[] bytes = FileManager.readFileIntoByteArray(m_serverfilename,true);
        String contentName = m_serverfilename.replace('/','_');
        // Build logical name out of file name, this logical name is
       // the default file name that is shown in the "save as" pop-up of the browser.
        // IMPORTANT: the extension must NOT be set, otherwise the browser opens
        // the document and does not offer the download pop-up
        contentName = contentName.replace('/','_');
        contentName = contentName.replace(':','_');
```

```
        contentName = contentName.replace('.','_');
       // pass to session buffer; use content type UNDEFINED,
       // not TEXT/HTML or some existing content type!
        m_downloadURL = findCISessionContext().getSessionBuffer().
        addDocument(contentName,bytes,"UNDEFINED");
        m_downloadURL = "<a target='_blank' href='" + m_downloadURL + ↩
"'>Download</a>";
        }
        catch (Throwable t)
        {
       outputMessage(MT_ERROR,"Could not load file from server! " + t.toString());
        m_downloadURL = null;
        }
    }
}
```

When the user defines the server file name, the following things happen:

- The property `serverFileName` is set in the adapter object.
- The file name is checked so that only files of a certain server directory are accessible for download.
- The file is read on server side from the file system.
- The content of the file is transferred to a so-called session buffer. The session buffer is a small technical framework that is available for all Application Designer applications: you can store any data inside the session buffer and assign a name. The session buffer returns a URL that allows browsers to access this content. (The URL internally contains a servlet call; the servlet is the one to pick the content when being accessed by a browser.)
- The URL that is returned from the session buffer is embedded into a `<a ...>...</a>` tag, representing a link to the user.
- The property `downloadURL` that is used in the IHTML control is the one to provide the URL.

Consequence: when changing the server file name in the example, the download link will be updated and the user can always download the corresponding file. The file name that is proposed in the **Save As** dialog box is the name that you assign to the session buffer content.

For proper content type management, you see that

- a content type was chosen (`UNDEFINED`) that is not defined and is consequently treated as “unknown content”,
- the name proposed for the download file does not contain an extension that is known to the browser (it does not contain an extension at all).

Properties

Basic			
valueprop	Server side property representation of the 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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
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.	Optional	1 2 3 4

	The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
ihtmlstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom

36 IMAGEOUT

The IMAGEOUT control is used to present images inside a page. The name of the image is not statically defined inside the layout but is taken from the value of an adapter property.

Example

XML layout definition:

```
<rowarea name="Image Out">
    <itr>
        <imageout valueprop="imageName">
        </imageout>
    </itr>
</rowarea>
```

Java code:

```
// property >imageName<
String m_imageName ="images/logo.gif";
public String getImageName() { return m_imageName; }
public void setImageName(String value) { m_imageName = value; }
```

The above layout definition, together with the above Java code, produces a page which looks as follows:

Loading Images from a Database, the File System, or Any Other Data Source

The previous example assumes that the image which is returned by the adapter program is located in such a way that the browser can reach it via a URL: the value "images/logo.gif" points to an image that is directly located inside the web application.

What can you do if the image is stored at a location that cannot be reached via a URL? Have a look at the following adapter program. It will produce exactly the same result as the previous example, but will explicitly load the image from the file system and then display it. Instead of loading the image from the file system, you could also load the image from a database or any other data source.

```
/** initialisation - called when creating this instance*/
public void init()
{
    // read image by file IO
    byte[] imageBytes = ↩
FileManager.readFileIntoByteArray("c:/temp/images/logo.gif",true);
    // add file to session buffer (name = TEST)
    SessionBuffer sb = findCISessionContext().getSessionBuffer();
    m_imageName = sb.addGIF("TEST",imageBytes);
}
```

You see that the image is read from the file system as a byte array and is then passed into a `SessionBuffer` object. This object is provided by Application Designer: you can store file content inside the buffer under a defined name. The session buffer returns a URL that allows the browser to access the file content. The session buffer stores the file in memory. It is bound to the user's session, i.e. it will be destroyed when the user ends the session. It also offers interfaces for removing content. You should remove content as quick as possible in order to save memory inside your application server.

See the Java API documentation for more information about the `SessionBuffer`.

Properties

Basic			
valueprop	Name of adapter property that provides as value the URL of the image that is shown inside the control. The URL must either be an absolute URL or a relative URL.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the 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 percentage 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%

height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	
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 synchronized 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	
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
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	

37 LABEL

The LABEL control is a static text. The tag has different properties to control the design of the label. It can be used to display plain text or as a headline of a grid.

By default, the label is rendered with a white line under the text. The default is suitable if a FIELD control follows the label.

Example

The XML layout definition is:

```
<rowarea name="Label Controls">
    <itr>
        <label name="Narrow" width="50">
        </label>
        <hdist>
        </hdist>
        <label name="Wide" width="150">
        </label>
        <hdist>
        </hdist>
        <label name="Plain" width="100" asplaintext="true">
        </label>
        <hdist>
        </hdist>
        <label name="Headline" width="100" asheadline="true">
        </label>
    </itr>
    <vdist>
    </vdist>
</rowarea>
```

For a better separation between the LABEL controls, horizontal distances (HDIST) were added.

Aligning the Text

Use the property `textalign` in order to align the label's text. Do not use the `align` property. `textalign` refers to the text inside the control, `align` refers to the position of the control inside the surrounding cell - if the cell is larger than the control.

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 percentage 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.	Sometimes 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	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
nowrap	If the textual content of the control exceeds the size of the control then the browser automatically breaks the line and arranges the text accordingly.	Optional	true false

	You can avoid this behaviour by setting NOWRAP to "true". No line break will be performed by the browser.		
width	(already explained above)		
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
asheadline	If set to true, the label has a dark background and the text is written in white (if using the standard style sheet). You may use this rendering style is you use labels as headlines of control grids (ROWTABLEAREA2 control).	Optional	true false
asplaintext	If set to true, no white line is drawn under the label text (if using the standard style sheet). You may use this rendering style if the label is used to name a RADIOBUTTON control or a CHECKBOX control.	Optional	true false
textalign	Horizontal alignment of the text that is shown.	Optional	left center right
cuttext	Boolean property defining the rendering if the text of the label does not fit into the defined width. If "true" then the text is cut - the part that does not fit is hidden. If "false" then the browser opens a second line. Default is "false".	Optional	true false
labelstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:	Optional	background-color: #FF0000 color: #0000FF

	border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		font-weight: bold
labelstyleclass	CSS style class used for rendering.	Optional	
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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2 VAR3 VAR4
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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	Optional	1 2 3

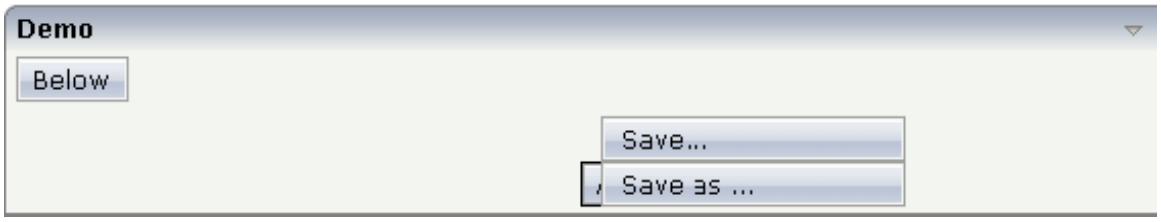
	default it is "1" - but you may want to define the control to span over more than one columns. The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		4 5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
Binding			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
nameprop	Name of adapter property that provides as value the text that is shown inside the control.	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

38 MENUBUTTON

The MENUBUTTON control offers the possibility to arrange buttons in a hierarchy.

Example

In the following example, there are two menu buttons which act differently when they are selected:



The XML code for the example looks as follows:

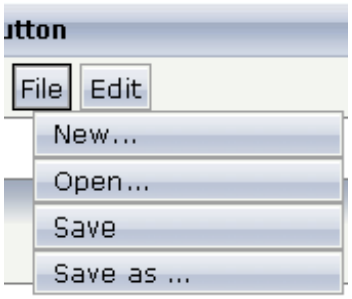
```
<rowarea name="Demo">
    <itr takefullwidth="true">
        <coltable0 width="50%" takefullheight="true">
            <itr>
                <menubutton name="Below" menuposition="below">
                    <menuitem name="New..." method="newFile" pixelwidth="150">
                    </menuitem>
                    <menuitem name="Open..." method="openFile" pixelwidth="150">
                    </menuitem>
                </menubutton>
            </itr>
        </coltable0>
        <coltable0 width="50%">
            <vdist height="50">
            </vdist>
            <itr>
```

```
                <menubutton name="Above" menuposition="above">
                    <menuitem name="Save..." method="saveFile" pixelwidth="150">
                    </menuitem>
                 <menuitem name="Save as ..." method="saveAsFile" pixelwidth="150">
                    </menuitem>
                </menubutton>
            </itr>
        </coltable0>
    </itr>
</rowarea>
```

In the definition of a menu item, a method of the adapter class is exactly referenced like a normal button.

Building a Button Menu

With the MENUBUTTON control, you can build simple menus:



Just place the MENUBUTTON controls inside the HEADER area.

MENUBUTTON Versus MENU

A complex MENU control is also available. The MENU control looks like a real menu, whereas the MENUBUTTON control is a special arrangement of normal buttons. See also the description of the **MENU** control in *Working with Menus*.

MENUBUTTON 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	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
menuposition	above if the menu should popup above the base menu button - below if the menu should popup below the base menu button. The default is below.	Optional	above below
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 percentage 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%
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
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.	Optional	1 2 3 4

	The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
buttonstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	

MENUITEM 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	

method	Method of your adapter object that is executed when clicking on the control.	Obligatory	
pixelwidth	Width of the control in pixels.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
pixelheight	Height of the control in pixels.	Optional	
itemstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	

39 METHODLINK

The METHODLINK is a control that renders a text that is dynamically derived from an adapter property. The text is rendered as a hyperlink. When clicking on the hyperlink, a method is called on the server side. It is used in scenarios in which users are in the habit of following links instead of choosing buttons or icons.

Example

The XML layout definition is:

```
<rowarea name="Method Link">
    <itr>
        <label name="Normal" width="100">
        </label>
        <methodlink method="onLink" valueprop="linkText" width="200">
        </methodlink>
    </itr>
    <itr>
        <label name="Straight Text" width="100">
        </label>
        <methodlink method="onLink" valueprop="linkText" width="300" ↩
straighttext="true">
        </methodlink>
    </itr>
</rowarea>
```

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.	Optional	
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".	Optional	
method	Method of your adapter object that is executed when clicking on the control.	Obligatory	
valueprop	Name of adapter property providing the text that is shown as link.	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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".	Optional	true false
linkstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

	Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
linkclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
nowrap	If the textual content of the control exceeds the size of the control then the browser automatically breaks the line and arranges the text accordingly. You can avoid this behaviour by setting NOWRAP to "true". No line break will be performed by the browser.	Optional	true false
colspan	Column spanning of control. If you use TR table rows then you may sometimes want to control the number of columns your	Optional	1 2

	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 synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		3 4 5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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			
valueprop	(already explained above)		
method	(already explained above)		
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
linkstatusprop	Name of the adapter property that dynamically passes information how the link should be rendered and how it should act. Valid values are "DISPLAY" and "EDIT".	Optional	
oncontextmenumethod	Name of the method on adapter level that is called when the user presses the right mouse button in an empty area.	Optional	
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	
pseudoclassessupport	Set this attribute to TRUE when you have set an own CSS style class for the LINKCLASS property which uses pseudo classes like ':hover' and ':link'. A HREF attribute will be created, which is required to make the pseudo classes work correctly. Set this property only if you use pseudo classes. Note: If you set this property the text "javascript:void(0)" will be displayed in the status bar when the mouse	Optional	true false

	pointer hovers over the link. This cannot be avoided.		

40 MULTISELECT

The MULTISELECT control allows comfortable input of multiple selections of items from a defined number of items.

Example

The available items are rendered on the left and are brought to the right by choosing the corresponding button. There are buttons to bring all items from the left to the right, and back.

The XML layout looks as follows:

```
<rowarea name="Control Demo">
    <itr>
        <label name="Multiselect Control" width="150">
        </label>
        <multiselect valueprop="towns" flush="server" width="300" ↩
helpid="MultiSelectHelp">
        </multiselect>
    </itr>
</rowarea>
```

The adapter code is:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.MULTISELECTInfo;
import com.softwareag.cis.server.util.OPTIONInfo;

// This class is a generated one.


public class MethodLinkAdapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------
```

```
    // property >towns<
    MULTISELECTInfo m_towns = new MULTISELECTInfo();
    public MULTISELECTInfo getTowns() { return m_towns; }

    // -------------------------------------------------------------------------
    // public access
    // -------------------------------------------------------------------------

    public void onOutput()
    {
        StringBuffer sb = new StringBuffer();
        OPTIONInfo[] items = m_towns.getItems();
        boolean first = true;
        for (int i=0; i<items.length; i++)
        {
            if (items[i].getSelected() == true)
        {
        if (first)
            first = false;
        else
            sb.append(", ");
        sb.append(items[i].getText());
        }
        }
        outputMessage(MT_SUCCESS,sb.toString());
    }

    // -------------------------------------------------------------------------
    // standard adapter methods
    // -------------------------------------------------------------------------

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_towns.addItem(new OPTIONInfo("Sevilla", "Sevilla", false));
        m_towns.addItem(new OPTIONInfo("Carmona", "Carmona", false));
        m_towns.addItem(new OPTIONInfo("Lebrija", "Lebrija", true));
        m_towns.addItem(new OPTIONInfo("Cadiz", "Cadiz", false));
        m_towns.addItem(new OPTIONInfo("Valencia", "Valencia", false));
        m_towns.addItem(new OPTIONInfo("Madrid", "Madrid", false));
        m_towns.addItem(new OPTIONInfo("Salamanca", "Salamanca", false));
        m_towns.addItem(new OPTIONInfo("Malaga", "Malaga", true));
        m_towns.addItem(new OPTIONInfo("Barcelona", "Barcelona", false));
        m_towns.addItem(new OPTIONInfo("Bilbao", "Bilbao", true));
        m_towns.addItem(new OPTIONInfo("Granada", "Granada", false));
    }
}
```

On the server side, the control is associated with an instance of class `MULTISELECTInfo`. The items are passed as `OPTIONInfo` objects. Depending on the boolean value inside the constructor, items

are either on the left side (unselected) or on the right side (selected). The `onOutput()` method shows how to derive information - which item was selected by the user.

Problems with MULTISELECT

With previous releases, the MULTISELECT control internally used the HTML control SELECT which has certain problems (see the description of the **COMBOFIX** control). The control is now rendered in a different way; the problems do not exist anymore.

Properties

Basic			
valueprop	Name of the adapter property representing this control on server side. The property must be of type MULTISELECTInfo. Please view corresponding documentation inside the Java API Documentation. The MULTISELECT control does not offer a STATUSPROP property in the way other controls (FIELD, ...) allow you to manipulate there input status at runtime. Instead you can set the status as method of the MULTISELECTInfo object that you create inside your adapter.	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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control	Obligatory	100 150 200

	(containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
withupdown	If set to true, corresponding up and down arrows appear on the right hand side. These arrows allow for changing the order of the selected items.	Optional	true false
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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.	Optional	1 2 3 4

	The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
msstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
Binding			
valueprop	(already explained above)		
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were	Optional	screen server

	changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.		
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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	

41 RADIOBUTTON

The RADIOBUTTON control displays the radio button. Radio buttons can be grouped together so that a group of RADIOBUTTON controls manipulates one property of an adapter control. Each RADIOBUTTON instance represents one value provided by the adapter property.

Example

The XML layout definition is:

```
<rowarea name="Radio Button">
    <itr>
        <radiobutton valueprop="option" value="1">
        </radiobutton>
        <label name="Option 1" asplaintext="true">
        </label>
    </itr>
    <itr>
        <radiobutton valueprop="option" value="2">
        </radiobutton>
        <label name="Option 2" asplaintext="true">
        </label>
    </itr>
</rowarea>
```

The Java code of the adapter is:

```
// property >option<
int m_option;
public int getOption() { return m_option; }
public void setOption(int value) { m_option = value; }
```

In the code example, an integer value is used as a property. You can also use any other kind of data type: string, boolean, float, etc.

Properties

Basic			
valueprop	Server side property representation of the control.	Obligatory	
value	Value that represents this instance of the RADIOBUTTON control. The value is set into the adapter property that is defined by the VALUEPROP property when the user clicks onto the control. - Vice versa: the control is switched to "marked" when the adapter property holds the value defined.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
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 percentage 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%
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because	Optional	left center right

	the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.		
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	xs:string ---------- N n.n P n.n string n
Label			
nameprop	Name of adapter property that provides as value the text that is shown inside the control.	Optional	
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.	Optional	
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".	Optional	
hdistpixelwidth	Width of the distance between checkbox and label in pixel.	Optional	
labelstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

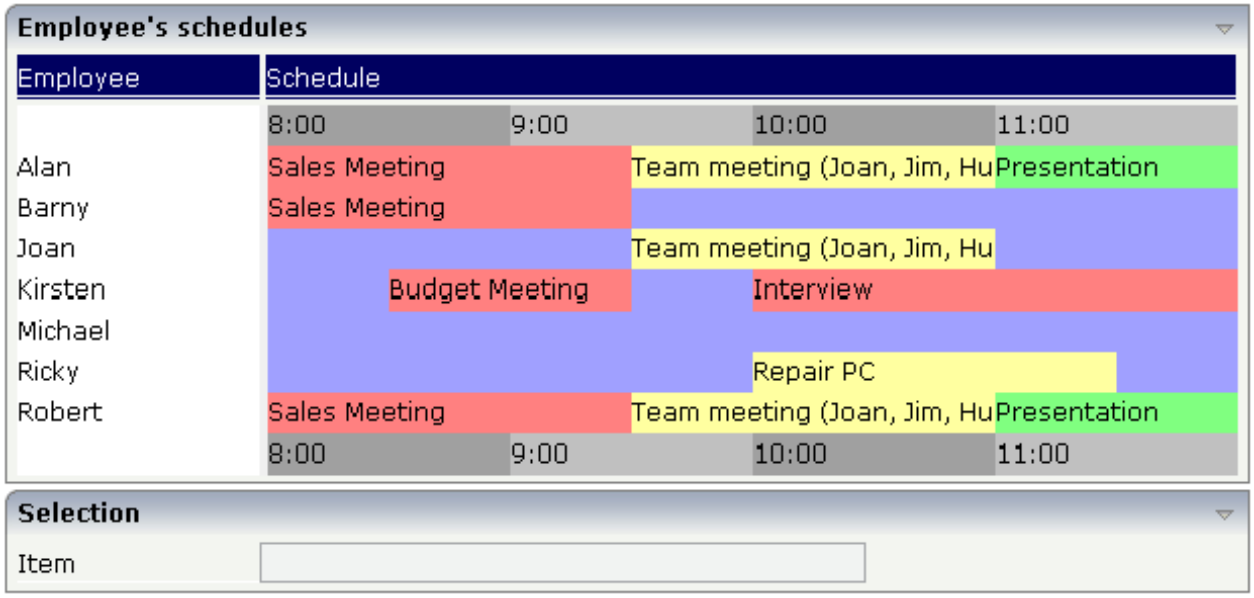
	are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
Binding			
valueprop	(already explained above)		
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
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 language management - representing the tooltip text that is used for the control.	Optional	
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	

The RADIOBUTTON control is typically followed by a label explaining its meaning.

42 SCHEDULELINE

The SCHEDULELINE control is used to define screens like the following:



You can display a certain sequence of items, each item holding a text, a color value, a size and an identifier. When clicking on an item, a certain method is called inside your adapter and the ID of the selected item is returned to perform activities in your program.

Example

The SCHEDULELINE control is very useful when used inside grids as shown in the above example screen. In principle, it is a standalone control that can (like any other control) be used inside a TABLEAREA grid. In this section, you find the explanation of the control. In *Working with Grids*, you will learn how to arrange single controls inside a grid.

First, have a look at a simple scenario in which the SCHEDULELINE control is used without a grid:

The XML layout definition looks as follows:

```
<rowarea name="Single Schedule Line">
    <itr>
        <label name="Schedule" width="120">
        </label>
        <scheduleline valueprop="schedule" width="450" pixelheight="20"
                      selectmethod="selectSchedule" selscheduleprop="selectedID"
                     seltypeprop="simpleSelType">
        </scheduleline>
    </itr>
    <itr>
        <label name="Selected Item" width="120">
        </label>
       <field valueprop="selectedID" flush="screen" length="3" displayonly="true">
        </field>
    </itr>
</rowarea>
```

The SCHEDULELINE definition links to a property `schedule` from which it derives the item information. If a selected method (`selectmethod` property) is called in the adapter, the ID of the selected item is passed into the property (`selschedule` property) before.

Let us have a look at the adapter code:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class Schedule_LineAdapter
    extends Adapter
{
    // property >schedule<
    String m_schedule;
    public String getSchedule() { return m_schedule; }
    public void setSchedule(String value) { m_schedule = value; }

    // property >selectedID<
    int m_selectedID;
    public int getSelectedID() { return m_selectedID; }
    public void setSelectedID(int value) { m_selectedID = value; }

/** */
    public void selectSchedule(){ }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        setSchedule("#FFa0a0;100;First Item;1;#A0FFA0;150;Second ↩
Item;2;#A0A0FF;200;Third Item;3");
    }
}
```

The most significant property is the `schedule` property. It derives its value from the class member `m_schedule`. The member is initialised with a string that represents the item information.

The string is structured in the following way:

- It contains values that are separated by semicolons, i.e. it follows the common “comma separated value” structure.
- Each item consists of four values, one after the other:
  - The color of the item in an HTML-understandable way.
  - The width of the item.
  - The text of the item - which can be blank.
  - The ID of the item. This ID can be blank. The control automatically changes the cursor no matter whether the item contains an ID or not. Items holding a key are selectable - items without a key are not selectable.
- If one of the four values is not available (e.g. no ID), it must not be left out. There must always be a sequence of four values. Example: if you want to display just an item with a color and a width value, this looks like "...;#FF0000;100;;;...".
- You can add as many items into the string as you desire.

The visible width of an item depends on the width of the SCHEDULELINE control itself (which is defined by the `width` property) and on the width value of the item. If the total width of a control is defined to be 100 pixels and each item ID is specified to get a width of "25%", then each item is actually getting 25% of the available 100 pixels.

Property `selectedId` receives the selected ID. Method `selectSchedule` is called whenever an item is selected. There is no implementation code in this example, but you could trigger e.g. page navigation or open a pop-up dialog on demand.

CSV Manager

Inside the Application Designer classes, there is the class `CSVManager` (package `com.softwareag.cis.file`) that supports the building of comma separated value strings. The class also covers the management of cases in which content parts of a CSV string themselves contain the separator character (conversion of ";" into "\;"). Documentation on this class is provided in the API JavaDoc documentation.

Use this class when encoding strings into CSV strings:

```
String csv = CSVManager.encodeString(new String[] {"1","2","3"});
```

Properties

Basic			
valueprop	Name of the adapter property representing the control's content on server side. The property must be of type "String". It returns a semicolon separated list of schedule items. Each item is represented by a color, a width, a text and a selection id. The width is not a pixel width but represents a "portion" that this schedule item represents. Example: #FF0000\:1000;Text 1;1;#00FF00;500;Text 2;2 The total "logical width" is 1500. The first item occupies 2/3 of the width, the right item occupies 1/3 of the width. The selection is required in case you want to react on user selections. If a user clicks onto one schedule item then the adapter is notified by a certain method - the id of the schedule item is passed as reference. Please have a look into the corresponding property descriptions.	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 percentage 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%
pixelheight	Height of the control in pixels.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Appearance			
width	(already explained above)		
pixelheight	(already explained above)		
pixelsizemode	A schedule line consists of sections, each one rendered with a certain width. By default the width does not represent a pixel value but represents a logical size. The width of the section depends on the logical size of one section compared with the logical size of the other sections. When switching this property to "true" then the size of the sections are interpreted as real pixel values.	Optional	true false
cellalign	Horizontal alignment of the text inside the control's schedule items.	Optional	left center right
cellvalign	Vertical alignment of the text inside the control's schedule items.	Optional	top middle bottom
cellstyle	Style that is used inside the schedule item cells. Can be any CSS style.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
cellnowrap	If switched to "true" then the text inside the schedule item cells is not broken if exceeding the size of the control - the text is cut instead. Default is "false".	Optional	true false
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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.	Optional	1 2 3 4

	The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
crosslineids	Flag (true | false) that indicates that cells of different lines (within ROWTABLEAREA2) does not have same ids. If set to false the control is able to detect and skip unnecessary re-draws (performance).	Optional	true false
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Binding			
valueprop	(already explained above)		
selectmethod	Adapter method that is called when the user selects one schedule item with the mouse.	Optional	
selscheduleprop	Name of adapter property in which the id of the selected schedule item is passed. The property is correctly set	Optional	

	before the method for reacting on the selection event is called.		
seltypeprop	Name of an adapter property that is used in the following way: If the user selects an item it can also be determined, if the item is selected by the left or by the right mouse button. In case the user uses the left mouse button, the value LEFT is passed into the property, which is referenced by the SELTYPEPROP property. In case the user uses the right mouse button, the value RIGHT is passed.	Optional	
preselectmode	If set to "true" then schedule items holding an id can be "preselected": the user can click on a schedule item and it is "grayed" as consequence - without directly calling the selection method. The selection method is called when double clicking onto the schedule item. Default is "false". The reaction of the control when clicking with the right mouse button remains untouched: still the selection method is called by a single right mouse button click.	Optional	true false
Vertical			
verticalschedule	Flag that indicates if the line is rendered vertically. Default is false.	Optional	true false
tooltipprop	Name of an adapter property of type "String" that contains the comma separated list of help texts that are displayed on mouse over (tooltip).	Optional	
imageprop	Name of the adapter property that returns a comma separated string of image URLs. An URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/green.gif;;red.gif"	Optional	
imageorientation	Flag that indicates to render the image at the left or right hand of the text.	Optional	left right
dropinfoprop	Name of an adapter property to that the id of the dragged cell is set. Do not use this property if you do not want to support drag and drop within the SCHEDULELINE. The server side property needs to be of type "String".	Optional	
onmovemethod	Name of an adapter method that is called on drop of one cell (source) over another cell (target). Use property DROPINFOPROP to get the id of the dragged cell	Optional	

	(source). Use SELSCHEDULEPROP to get the id of the cell that got the drop (target).		
controlkeyprop	Name of an adapter property to that the information is set whether the user pressed the CTRL key when selecting a cell. Property needs to be of type "boolean".	Optional	

43 SLIDER

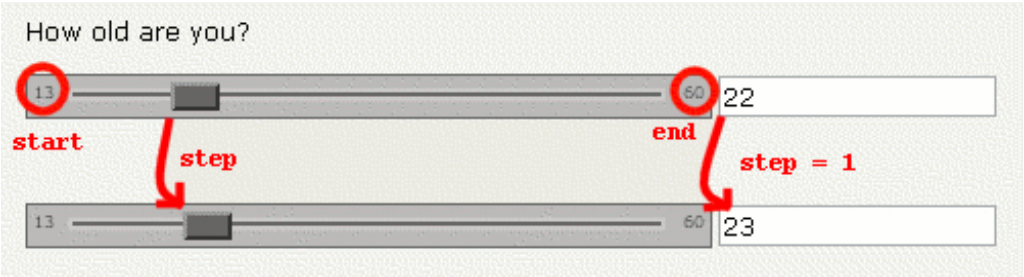
The SLIDER control represents a slider. The main use of the slider is to limit the user input to specific values. It uses a number representation for its values, but the numbers can also be used to express string values.

Example

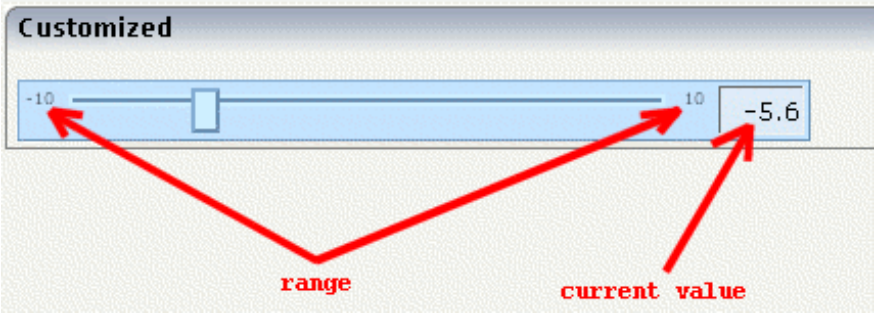
The XML layout definition is:

```
<rowarea name="Number Output">
    <itr>
        <slider valueprop="slider1" from="13" to="60" showrange="true"
                                                showcurrentvalue="false">
        </slider>
    </itr>
</rowarea>
```

The control can be customized by setting its start value, end value and a step. The start and end values form a closed interval. The step defines the distance between two valid values represented by the slider in this interval.



In the above example, the value for the step is the default value "1". The possible values represented by the slider are the integers from "13" to "60". It is possible to specify a floating-point number as a step, for example "0,25". The slider can be further customized by setting the properties `showrange` and `showcurrentvalue` which show the range (start and end value) and the current value of the slider while the user is moving it. The width and height of the slider point is adjustable. The slider point is the element which the user drags and drops. The colors, the borders of the slider, the point, the line, the range and the current value can also be customized.



The Java code of the adapter is:

```
package com.softwareag.cis.test22;
// This class is a generated one.

import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;

public class SliderAdapter
    extends Adapter
{

    public class NumberSLIDERInfo extends SLIDERInfo
    {
        public void trigger()
        {
            m_fieldNumber = this.giveFormattedSliderValueAsInt();
        }
    }

    public class StringSLIDERInfo extends SLIDERInfo
    {
        public void trigger()
        {
         int inpValue = this.giveFormattedSliderValueAsInt();
         if(inpValue == 1) m_fieldString = "very bad";
         else if(inpValue == 2) m_fieldString = "bad";
         else if(inpValue == 3) m_fieldString = "ok";
         else if(inpValue == 4) m_fieldString = "good";
         else if(inpValue == 5) m_fieldString = "very good";
        }
    }

    // property >fieldNumber<
    int m_fieldNumber;
    public int getFieldNumber() { return m_fieldNumber; }
    public void setFieldNumber(int value)
    {
        m_fieldNumber = value;
    }
```

```
    // property >slider1<
    NumberSLIDERInfo m_slider1=new NumberSLIDERInfo();
    public NumberSLIDERInfo getSlider1() { return m_slider1; }
    public void setSlider1(NumberSLIDERInfo value) { m_slider1 = value; }

    // property >fieldString<
    String m_fieldString;
    public String getFieldString() { return m_fieldString; }
    public void setFieldString(String value) { m_fieldString = value; }

    // property >slider2<
    StringSLIDERInfo m_slider2=new StringSLIDERInfo();
    public StringSLIDERInfo getSlider2() { return m_slider2; }
    public void setSlider2(StringSLIDERInfo value) { m_slider2 = value; }

    // property >slider3<
    SLIDERInfo m_slider3=new SLIDERInfo();
    public SLIDERInfo getSlider3() { return m_slider3; }
    public void setSlider3(SLIDERInfo value) { m_slider3 = value; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_slider1.setSliderValue(18);
        m_slider2.setSliderValue(3);
    }

    /** */
    public void onEnterNumber()
    {
        m_slider1.setSliderValue(m_fieldNumber);
        m_fieldNumber = m_slider1.giveFormattedSliderValueAsInt();
    }

}
```

Every slider is bound to a `SLIDERInfo` object. Calling the `setSliderValue()` method sets the value of the slider which automatically moves it. However, the slider is not set to every given value, but to one valid value which fits in the given interval. The valid value to which the slider is currently set can be obtained by calling the `giveFormattedSliderValueAsInt()` or `giveFormattedSliderValueAsFloat()` method. When the user drops the slider, the method `trigger()` is called. This can be used to define specific behavior when the value in the user interface is changed. As in the example above, the `SLIDERInfo` can be extended, and the `trigger()` method can be overwritten to define specific behavior. You can show the new value in another control or use it for other purposes.

Properties

Basic			
valueprop	Server side property representation of the control.	Obligatory	
Appearance			
width	Width of the slider. Can be given in pixels or percentage.	Optional	100 120 140 160 180 200 50% 100%
displayonly	If set to true, the SLIDER will not be accessible for input. It is just used as an output.	Optional	true false
showrange	Boolean value. Whether to show the range of the slider. The range is the "from" and "to" values.	Optional	true false
showcurrentvalue	Boolean value. Whether to show the current value of the slider while it is moving.	Optional	true false
mainbgcolor	Background color of the slider container. This should be a valid CSS color value. For example a name(blue, red), a hexadecimal value(#99CCFF) or others.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
mainbordercolor	Border color of the slider container.	Optional	#bbb #666 #666 #bbb

	This should be a valid CSS border-color value. You can specify a different color for the top, right, bottom and left border in this sequence. For example: #BBBBBB #666666 #666666 #BBBBBB		#BFCFFF #00248F #00248F #BFCFFF
mainborderwidth	Border width of the slider container.	Optional	thin medium thick 1px 2px 5px 10px
pointbgcolor	Background color of the slider point. This should be a valid CSS color value. For example a name(blue, red), a hexadecimal value(#99CCFF) or others.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
pointbordercolor	Border color of the slider point. This should be a valid CSS border-color value. You can specify a different color for the top, right, bottom and left border in this sequence. For example: #BBBBBB #666666 #666666 #BBBBBB	Optional	#bbb #666 #666 #bbb #BFCFFF #00248F #00248F #BFCFFF
pointborderwidth	Border width of the slider point.	Optional	thin medium thick 1px 2px 5px 10px

pointwidth	Width of the slider point in pixels. The value must be an integer value.	Optional	10 20 40 100 300
pointheight	Height of the slider point in pixels. The value must be an integer value.	Optional	10 20 40 100 300
linebgcolor	Background color of the slider line. This should be a valid CSS color value. For example a name(blue, red), a hexadecimal value(#99CCFF) or others.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
lineborderc olor	Border color of the slider line. This should be a valid CSS border-color value. You can specify a different color for the top, right, bottom and left border in this sequence. For example: #BBBBBB #666666 #666666 #BBBBBB	Optional	#bbb #666 #666 #bbb #BFCFFF #00248F #00248F #BFCFFF
lineborderwidth	Border width of the slider line.	Optional	thin medium thick 1px 2px 5px 10px

rangefontsize	Font size of the slider range.	Optional	xx-small x-small small medium large x-large xx-large smaller larger 150%
valuebgcolor	Background color of the slider current value which is shown if the "showcurrentvalue" property is set to true. This should be a valid CSS color value. For example a name(blue, red), a hexadecimal value(#99CCFF) or others.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
valueborderc olor	Background color of the slider current value which is shown if the "showcurrentvalue" property is set to true. This should be a valid CSS border-color value. You can specify a different color for the top, right, bottom and left border in this sequence. For example: #bbb #666 #666 #bbb	Optional	#bbb #666 #666 #bbb #BFCFFF #00248F #00248F #BFCFFF
valueborderwidth	Border width of the slider current value which is shown if the "showcurrentvalue" property is set to true.	Optional	thin medium thick 1px 2px 5px 10px

valuefontsize	Font size of the slider current value which is shown if the "showcurrentvalue" property is set to true.	Optional	xx-small x-small small medium large x-large xx-large smaller larger 150%

44 STRIPSEL

The STRIPSEL control is very similar to the TABSTRIP2 control: the user selects one option out of many.

The STRIPSEL control is typically located somewhere at the top of a page, but it can also be positioned anywhere else.

Example

Programming a STRIPSEL control is the same as programming the TABSTRIP2 control - just the rendering of the control differs:

In this example, the STRIPSEL control is the control below the titlebar. For comparison, the TABSTRIP2 control has also been added.

The XML layout is:

```
<page model="com.softwareag.cis.test40.StripselAdapter">
    <titlebar name="STRIPSEL Control">
    </titlebar>
    <stripsel tabstripprop="stripsel">
    </stripsel>
    <pagebody>
        <rowarea name="Test">
            <itr>
                <label name="Selection" width="120">
                </label>
                <field valueprop="selection" width="200" displayonly="true">
                </field>
            </itr>
        </rowarea>
        <rowarea name="Comparison with TABSTRIP Control">
            <tabstrip2 tabstripprop="stripsel">
            </tabstrip2>
        </rowarea>
        ...
        ...
```

```
        ...
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The Java code of the adapter is:

```
package com.softwareag.cis.test40;

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.test30.Tabstrip2Adapter.MyTABSTRIPInfo;
import com.softwareag.cis.util.*;

public class StripselAdapter
    extends Adapter
{
    public class MyTABSTRIPInfo
        extends TABSTRIPInfo
    {
        public void reactOnTabSelection(int index)
        {
            m_selection = this.getItems()[index].getName();
        }
    }

    String m_selection;
    public String getSelection() { return m_selection; }
    public void setSelection(String value) { m_selection = value; }

    TABSTRIPInfo m_stripsel = new MyTABSTRIPInfo();
    public TABSTRIPInfo getStripsel() { return m_stripsel; }

    public void init()
    {
        m_stripsel.addItem("First");
        m_stripsel.addItem("Second");
        m_stripsel.addItem("Third");
        m_stripsel.addItem("Fourth");
        m_stripsel.selectTab(0);
    }

}
```

Properties

Basic			
tabstripprop	Name of the adapter property that represents the control on server side. The property must be a subclass of "TABSTRIPInfo". In your implementation you must override method "reactOnTabSelection(index)".	Optional	
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
scrollable	Flag that indicates if the control shows scroll icons on the right upper corner. Default is true	Optional	true false
backgroundstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

scrolllefttitle	Help text that is displayed if the user moves the mouse of the scroll to left icon.	Optional	
scrolllefttitletextid	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".	Optional	
scrollrighttitle	Help text that is displayed if the user moves the mouse of the scroll to right icon.	Optional	
scrollrighttitletextid	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".	Optional	
scrollleftimage	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. specifying "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	gif jpg jpeg
scrollleftimagertl	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. specifying "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	gif jpg jpeg
scrollrightimage	URL of image that is displayed inside the control. Any image type (.gif, .jpg, ...) that your browser does understand is valid.	Optional	gif jpg

	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. specifying "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".		jpeg
scrollrightimagertl	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. specifying "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	gif jpg jpeg
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

45 SUBPAGE

The SUBPAGE control defines an area in which an HTML page is shown. The URL of the page is not statically defined, but is derived from a value of an adapter property.

Due to the browser's capability to embed installed plug-ins, you can use non-HTML objects to be called - and which the browser is able to understand. For example, if you have Microsoft Office installed (or the viewers for Microsoft Office documents) and you pass the name of a Word document as the URL, the Word document will be embedded into the page.

Example

```
<rowarea name="Subpage">
    <itr>
        <label name="URL" width="100">
        </label>
        <field valueprop="url" flush="true" length="30">
        </field>
    </itr>
    <vdist>
    </vdist>
    <vdist>
    </vdist>
    <itr>
        <subpage valueprop="url" height="300" width="400">
        </subpage>
    </itr>
</rowarea>
```

The above XML layout definition produces a page which looks as follows:

Typical Problem: Non-Refreshing Subpages

Sometimes the SUBPAGE control is used to embed a generated HTML page into an existing page: the adapter program somehow creates an HTML document (e.g. a certain list output) and saves this output to the file system in such a way that it can be reached via a URL. This URL is passed back as a value into the SUBPAGE control.

Pay attention to the fact that the SUBPAGE control only refreshes its content if the URL changes. If the URL stays constant, the SUBPAGE control does refresh its inner content. Consequence: if you have situations in which the user stays on the same page and creates new subpages multiple times, these subpages must have a URL which changes on every new creation.

A way to force this reloading of a subpage is to append a parameter to the URL which changes every time you want to refresh. Example: if the page's URL is *http://xyz.xyz.html*, then you can write your adapter program in the following way:

```
public void getPageURL()
{
    return m_pageURL;
}

public void onReloadPageURL()
{
    m_pageURL = "http://xyz.xyz.html?DUMMPARAM=" + (new Date()).getTime();
}
```

Check the reload behavior with your web server first.

Properties

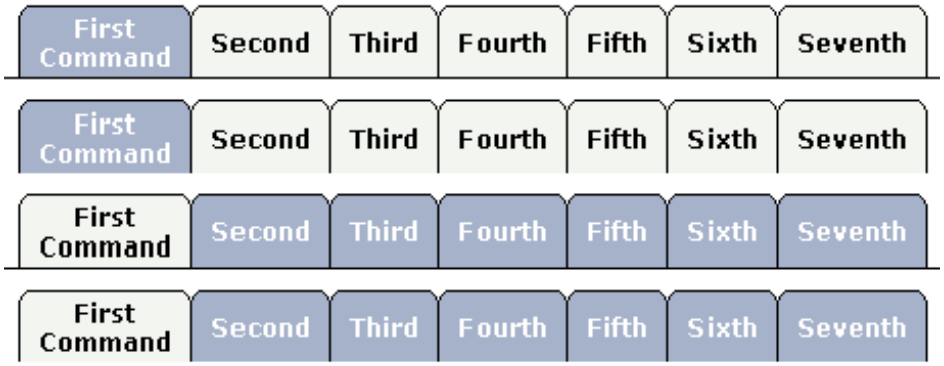
Basic			
valueprop	Name of adapter property providing the URL to be displayed inside the SUBPAGE control. The URL may either be an absolute URL ("http://....) or may be a relative URL. Please note: the SUBPAGE control only re-renders its inner content if the URL provided by the property really changes. The SUBPAGE control does not "know" if something changed inside the contained page and that it has to redraw the page. - If you want to refresh the inner page explicitly append some random number to your URL, e.g.: http://...url...?RANDOM=45435. By changing the number the browser will relaod the URL.	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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%

height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Sometimes obligatory	100 150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
height	(already explained above)		
scrolling	Definition of the scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Optional	auto yes no
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
colspan	Column spanning of control.	Optional	1

	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 synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		2 3 4 5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
alwaysreload	When setting to false, the subpage is not reloaded when a page switch is executed, default is true.	Optional	true false
Binding			
valueprop	(already explained above)		

46 TABSEL

The TABSEL control looks as shown in the following example:



The number of tabs is dynamically defined at runtime. There are various output options:

- With/without a horizontal line below the control.
- Normal or reverse coloring.

Like the TABSTRIP control, the TABSEL control does not provide internal containers that are switched when selecting tabs. It just represents one tab line.

Example

The XML layout of above example is:

```
<pagebody horizdist="false">
    <tabsel tabselprop="tabsel">
    </tabsel>
    <vdist height="10">
    </vdist>
    <tabsel tabselprop="tabsel" bottomborder="false">
    </tabsel>
    <vdist height="10">
    </vdist>
    <tabsel tabselprop="tabsel" bottomborder="true" reversecolors="true">
    </tabsel>
    <vdist height="10">
    </vdist>
    <tabsel tabselprop="tabsel" bottomborder="false" reversecolors="true">
    </tabsel>
</pagebody>
```

The adapter code is:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TABSELInfo;

// This class is a generated one.

public class TabSelAdapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // inner classes
    // -----------------------------------------------------------------------

    // class >TabselInfo<
    public class TabselInfo extends TABSELInfo
    {
        public void reactOnSelect()
        {
            outputMessage(MT_SUCCESS,"Selected: " + getSelectedItem());
        }
    }

    // -----------------------------------------------------------------------
    // properties
    // -----------------------------------------------------------------------

    // property >tabsel<
    TabselInfo m_tabsel = new TabselInfo();
    public TabselInfo getTabsel() { return m_tabsel; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        // Fill TABSEL
        m_tabsel.exchangeItems(new String[]
                                        {
                                        "First Command",
                                        "Second",
                                        "Third",
                                        "Fourth",
                                        "Fifth",
                                        "Sixth",
                                        "Seventh"
                                        });
    }
}
```

On the server side, the control is associated with an instance of class `MyTABSELInfo` - derived from `TABSELInfo`. The instance is loaded with the available tabs. When the user selects a tab, a method `reactOnSelect` is called inside the instance. The program can access the selected item by using `getSelectedItem()` - returning the index of the item selected.

Properties

Basic			
tabselprop	Name of the adapter property representing the TABSEL control on server side. The property must be of type "TABSELInfo". Please find further information inside the Java API Documentation.	Obligatory	
bottomborder	If set to "true" then a bottom border is rendered below the tab selection. If set to "false" then no bottom border will be drawn.	Optional	true false
reversecolors	Reverses the color scheme of the TABSEL control.	Optional	true false
leftindent	Inserts a horizontal distance left of the first "tab" and shifts the "tabs" to the right as consequence. The value you may define represents the number of pixels that are inserted.	Optional	1 2 3 int-value
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

47 TABSTRIP2

The TABSTRIP2 control is used to navigate through certain aspects of your application. The way you navigate depends completely on your implementation.

Note: TABSTRIP2 is a wrapper around the TABSTRIP control which was available with previous releases. In the TABSTRIP control, you had to do quite a lot of references to adapter properties that are now replaced by one TABSTRIPInfo object. The TABSTRIP control is still supported.

Example

The control looks as follows:

For each aspect, there is one tab holding a name and an index. The left-most tab holds index 0, the next one 1, etc.

The XML layout definition is:

```
<page model="tabstripAdapter">
    <titlebar name="Simple programming of tab-strip-controls">
    </titlebar>
    <tabstrip2 tabstripprop="titletab" scrollable="false">
    </tabstrip2>
    <header withdistance="false">
        <button name="Exit">
        </button>
    </header>
    <pagebody>
        <rowarea name="Selection">
            <itr>
                <label name="Selected tab" width="120">
                </label>
                <field valueprop="selTabText">
                </field>
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
```

```
    </statusbar>
</page>
```

Note: Whereas the ROWTABAREA and COLTABAREA controls explicitly open one container (TABPAGE) per tab, the TABSTRIP2 control is just a "tab line" itself without any container included.

The adapter class looks as follows:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TABSTRIPInfo;

// This class is a generated one.

public class tabstripAdapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // inner classes
    // -----------------------------------------------------------------------

    public class TitleTABSTRIPInfo
    extends TABSTRIPInfo
    {
        public void reactOnTabSelection(int index)
        {
            m_selTabText = this.findSelectedItem().getName();

        }
    }

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

    // property >selTabText<
    String m_selTabText;
    public String getSelTabText() { return m_selTabText; }
    public void setSelTabText(String value) { m_selTabText = value; }

    // property >titletab<
    TABSTRIPInfo m_titletab = new TitleTABSTRIPInfo();
    public TABSTRIPInfo getTitletab() { return m_titletab; }
    public void setTitletab(TABSTRIPInfo value) { m_titletab = value; }

    // -----------------------------------------------------------------------
    // standard adapter methods
    // -----------------------------------------------------------------------

    /** initialisation - called when creating this instance*/
    public void init()
```

```
    {
        m_titletab.addItem("First"); // tab 0
        m_titletab.addItem("Second"); // tab 1
        m_titletab.addItem("Third"); // tab 2
        m_titletab.selectTab(0);
    }
}
```

The TABSTRIP2 control is represented by a `TABSTRIPInfo` object on the server side. You see that a subclass of its own is generated (`TitleTABSTRIPInfo`) in which the method `reactOnTabSelection(...)` is overwritten. This method is called when the user selects a tab.

The `TitleTABSTRIPInfo` object is initialised inside the `init()` method: there are corresponding methods for adding items and for selecting the one that represents the currently selected one. You can change the information behind this (e.g. the number of tabs to be shown) at any time - also at a later point in time, after initialisation. But: do not create new instances of the object, use the `clear()` method instead.

TABSTRIP2 - Usage with Other Controls

The TABSTRIP2 control may not only be used below the titlebar. It can also be arranged on top or below many other controls. The following example shows the usage together with a ROWAREA control:

The XML layout definition is:

```
<rowarea name="Info" height="200">
</rowarea>
<tabstrip2 tabstripprop="tabstrip" backgroundstyle="background-color: #FFFFFF">
</tabstrip2>
```

You see that the background style of the TABSTRIP2 control was explicitly set to white in order to fit into the coloring of the ROWAREA.

Properties

Basic			
tabstripprop	Name of the adapter property that represents the control on server side. The property must be a subclass of "TABSTRIPInfo". In your implementation you must override method "reactOnTabSelection(index)".	Optional	
align	Horizontal alignment of the control's content. Default is "center".	Optional	left center right
scrollable	If set to "true" then small icons will appear on the right border of the control. If the size of the "tabs" is too big and some tabs are cut as consequence then you can use these icons for scrolling left and right.	Optional	true false
backgroundstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

scrollleftimage	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. specifying "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	gif jpg jpeg
scrollleftimagertl	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. specifying "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	gif jpg jpeg
scrollrightimage	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. specifying "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	gif jpg jpeg
scrollrightimagertl	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. specifying "images/xyz.gif" will point into a directory parallel to your	Optional	gif jpg jpeg

	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".		

48 TAGCLOUD

The TAGCLOUD control represents a collection of tags. A tag is a keyword assigned to an information resource (picture, video clip or others). In a tag cloud, the tags are mainly shown by their popularity.

Example

As you can see, different tags can be added to a tag cloud. They differ by their popularity. The most popular tags are those with a bigger font size.

The XML layout definition is:

```
<itr>
   <tagcloud tagcloudprop="tagCloud"
          width="300" height="350"
          borderstyle="dotted" borderwidth="1px"
          bordercolor="#0000FF" backgroundcolor="#E6E6FA"
          textcolor="#0000FF">
   </tagcloud>
</itr>
```

The tag cloud can be customized by defining a background color. Adapt the look and feel to your own style (see *Adapting the Look & Feel*).

The Java code of the adapter is:

```
public classTagcloudAdapter
    extends Adapter
{
    /** sub class TAGCLOUDInfo to react on click events*/
    public class MyTagcloudInfo extends TAGCLOUDInfo
    {
      public void trigger()
      {
        outputMessage(MT_SUCCESS, getSelectedTagName());
      }
    }

    // property >tagCloud<
    MyTagcloudInfo m_tagCloud=new MyTagcloudInfo();
    public MyTagcloudInfo getTagCloud() { return m_tagCloud; }

    /** called on page load*/
    public void init()
    {
        m_tagCloud.addTag("computer");
        m_tagCloud.addTag("technology");
        m_tagCloud.addTag("java", TAGCLOUDInfo.TAGPOPULARITY_5_VERYBIG);
        m_tagCloud.addTag("books");
        m_tagCloud.addTag("drinks");
        m_tagCloud.addTag("music", TAGCLOUDInfo.TAGPOPULARITY_4_BIG);
        m_tagCloud.addTag("people");
        m_tagCloud.addTag("germany", TAGCLOUDInfo.TAGPOPULARITY_4_BIG);
        m_tagCloud.addTag("summer", TAGCLOUDInfo.TAGPOPULARITY_4_BIG);
        m_tagCloud.addTag("flowers");
        m_tagCloud.addTag("kids", TAGCLOUDInfo.TAGPOPULARITY_2_SMALL);
        m_tagCloud.addTag("holiday");
        m_tagCloud.addTag("semantic", TAGCLOUDInfo.TAGPOPULARITY_4_BIG);
        m_tagCloud.addTag("micro", TAGCLOUDInfo.TAGPOPULARITY_2_SMALL);
        m_tagCloud.addTag("birthday");

        . . .

        . . .
    }
}
```

A tag cloud is bound to a `TAGCLOUDInfo` object. Using this object you can add tags with different popularities. All tags with the same popularity have their own stylesheet class. You can also remove tags and get the last tag which has been selected by the user. When you want to react on click events, you can override the `trigger()` function of the `TAGCLOUDInfo` object and add specific code.

Properties

Basic			
tagcloudprop	Name of the adapter property that represents the control on server side. Return type must be "TAGCLOUDInfo".	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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
borderstyle	Choose the style the controls border.	Optional	solid double groove

			dotted dashed inset outset ridge hidden
borderwidth	Border size of control in pixels. Specify "0" not to render any border at all.	Optional	thin medium thick 1px 2px 5px 10px
bordercolor	Sets the border color of the control.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
backgroundcolor	Sets the background color of the control.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
textcolor	Sets the text color of the control.	Optional	#FF0000 #00FF00

			#0000FF #FFFFFF #808080 #000000

49 TEXT

The TEXT control represents a multi line text edit control. It represents the value of an adapter property.

Example

The XML layout definition is:

```
<rowarea name="Text">
    <itr>
        <text valueprop="longText" rows="7" cols="30">
        </text>
    </itr>
</rowarea>
```

Using the max* Properties

This section explains when and how to use the following properties:

`maxlength/maxlengthprop`
`maxrows/maxrowsprop`
`maxrowlength/maxrowlengthprop`

The rule of thumb is: `maxlength/maxlengthprop` supports a completely different use case than `maxrows/maxrowsprop` in combination with `maxrowlength/maxrowlengthprop`. Never use all three.

If you simply want to limit the total number of characters that your end users are allowed to enter, use `maxlength/maxlengthprop`.

If you prefer that your end users enter their input line by line with a fixed line length and no automatic word wrapping, use `maxrows/maxrowsprop` in combination with `maxrowlength/maxrowlengthprop`.

Properties

Basic			
valueprop	Server side property representation of the control.	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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour.	Optional	screen server

	Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.		
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	string n xs:string
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
direction	Presets the default(BiDi) direction of the control. Use black string in order to have the default value.	Optional	rtl ltr
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	

statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
wrap	Specifies the line wrapping inside the control. By default a line that exceeds the width of the control is broken automatically. You may define this property to not wrap at all ("off") - in this case the text control offers horizontal scroll bars to scroll the text. There are two styles of wrapping "soft" and "hard". The difference between "soft" and "hard" is the way the text is - if changed by the user - passed back to the adapter property: when specifying "soft" then line breaks which are caused by wrapping are not sent to the server, when specifying "hard" then line breaks caused by wrapping are sent as carriage return/ line feed. - Be careful when specifying "hard" as consequence! The wrap attribute is not part of the HTML standard. It depends on the browser if wrap=hard/soft are supported.	Optional	soft hard off
rows	Height of control specified by number of rows. Either define the height by the HEIGHT property or by the ROWS property. Do not specify both! When specifying the height by ROWS then be aware of that the height depends from the font size used inside the control (that is defined in the styles sheet definition).	Optional	
cols	Width of control specified by number of characters. Either define the width by the WIDTH property or by the COLS property. Do not specify both! When specifying the width by COLS then be aware of that the width depends from the font size used inside the control (that is defined in the styles sheet definition).	Optional	
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 synchronized 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
maxlength	Maximum number of characters that a user may enter. This property is not depending on the LENGTH property - please do not get confused by the similar naming. MAXLENGTH has	Optional	5 10

	nothing to do with the optical sizing of the control but only with the number of characters you may input.		15 20 int-value
maxlengthprop	Name of adapter property that passes back the maximum number of characters that a user may enter. Consider to use MAXLENGTH to define this number in a static way.	Optional	
maxrows	Maximum number of rows. No automatic wrapping is done. Don't use this in combination with maxlength/maxlengthprop.	Optional	20 50 100 200 500 0
maxrowsprop	Name of the adapter property that dynamically sets the maximum number of rows.	Optional	
maxrowlength	Maximum number of characters in a row. No automatic wrapping is done. Don't this in combination with maxlength/maxlengthprop.	Optional	5 10 15 20 int-value
maxrowlengthprop	Name of the adapter property that dynamically sets the maximum number of characters in a row.	Optional	
autowordwrap	Can only be used in combination with maxrowlength/maxrowlengthprop and maxrows/maxrowsprop. If set to "true" than when maxrowlength/maxrowlengthprop is reached an automatic word wrap to the next line will be done.	Optional	true false
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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

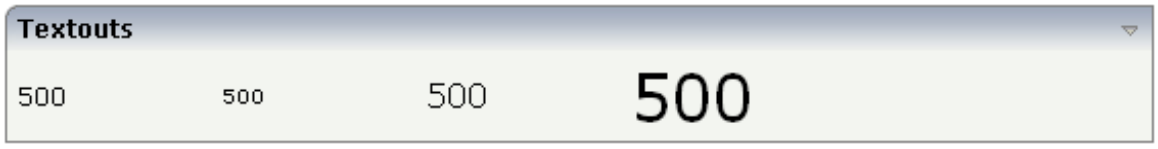
			int-value
textareastyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
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 language management - representing the tooltip text that is used for the control.	Optional	
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
bgcolorprop	Name of adapter property that passes back a color value (e.g. "#FF0000" for red color). The color value is used as background color in the control. - The color of the text color is automatically chosen dependent from the background color: for light background colors the text color is black, for dark background colors the color is white. Use FGCOLORPROP to choose the text color on your own.	Optional	
fgcolorprop	Name of adapter property that passes back a color value (e.g. "#FF0000" for red color). The color value is used as text color in the control. - The background color is automatically chosen dependent from the text color: for dark text colors the background color is transparent (default), for light text colors the color is black. Use BGCOLORPROP to choose both - the text and background color.	Optional	
scroll	Definition of the scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden").	Optional	auto scroll hidden

	Default is "auto".		
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
Binding			
displayprop	(already explained above)		
statusprop	(already explained above)		
titleprop	(already explained above)		
bgcolorprop	(already explained above)		
fgcolorprop	(already explained above)		
maxlengthprop	(already explained above)		
maxrowsprop	(already explained above)		
maxrowlengthprop	(already explained above)		
Online Help			
helpid	Help id that is passed to the online help management in case the user presses F1 on the control.	Optional	
title	(already explained above)		
titletextid	(already explained above)		
titleprop	(already explained above)		
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	

50 TEXTOUT

The TEXTOUT control is used to display plain text. The text is not statically defined (as a label) but is derived from a property of the adapter class.

Example



The XML layout definition is:

```
<rowarea name="Textouts">
    <itr>
        <textout valueprop="factor1" width="100">
        </textout>
        <textout valueprop="factor1" width="100" textsize="1">
        </textout>
        <textout valueprop="factor1" width="100" textsize="3">
        </textout>
        <textout valueprop="factor1" width="100" textsize="6">
        </textout>
    </itr>
</rowarea>
```

Example: Dynamic Labels

By using the `styleclass` property of the TEXTOUT control, you can define text output that looks like a normal LABEL control. However, instead of a fixed text, it has a text that is dynamically derived from the adapter logic:

The layout definitions is:

```
<rowarea name="Text">
    <itr>
        <textout valueprop="dynprop" width="120" textoutclass="LABELCellNormal">
        </textout>
        <field valueprop="dynlabel" width="200">
        </field>
    </itr>
</rowarea>
```

In the above example, the left **First Name** is not a label but a TEXTOUT control, referencing to the style class `LABELCellNormal` that normally is a style class belonging to the LABEL control.

Example: Dynamic Labels with Tooltips

By extending the previous example, you can also add tooltips to the dynamic label:

The implementation of the adapter property is:

```
// property >dynlabel<
String m_dynlabel ="Harald";
public String getDynlabel() { return m_dynlabel; }
public void setDynlabel(String value) { m_dynlabel = value; }
```

The text of the value that is passed back is encapsulated within an HTML span. The span itself provides the property `title`.

Properties

Basic			
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").	Sometimes obligatory	100 120 140 160 180

	(C) Percentage sizing: input a percentage 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.		200 50% 100%
valueprop	Server side property representation of the control.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
nowrap	If the textual content of the control exceeds the size of the control then the browser automatically breaks the line and arranges the text accordingly. You can avoid this behaviour by setting NOWRAP to "true". No line break will be performed by the browser.	Optional	true false
textsize	The HTML font size of the text. Corresponding to the HTML definition "1" means "smallest" and "6" means "biggest".	Optional	1 2 3 4 5 6

textcolor	Colour of the text. Input a value like "#FF0000".	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	date float int long time timestamp color xs:decimal xs:double xs:date xs:dateTime xs:time ---------- N n.n P n.n string n L xs:boolean xs:byte xs:short
straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying	Optional	true false

	STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".		
align	Horizontal alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control itself. In this case the "align" property specifies the position of the control inside the column. In most cases you do not require the align control to be explicitly defined because the size of the column around the controls exactly is sized in the same way as the contained control. If you want to directly control the alignment of text: in most text based controls there is an explicit property "textalign" in which you align the control's contained text.	Optional	left center right
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
bgcolorprop	Name of adapter property that passes back a color value (e.g. "#FF0000" for red color). The color value is used as background color in the control. - The color of the text color is automatically chosen dependent from the background color: for light background colors the text color is black, for dark background colors the color is white. Use FGCOLORPROP to choose the text color on your own.	Optional	
fgcolorprop	Name of adapter property that passes back a color value (e.g. "#FF0000" for red color). The color value is used as text color in the control. - The background color is automatically chosen dependent from the text color: for dark text colors the background color is transparent (default), for light text colors the color is black. Use BGCOLORPROP to choose both - the text and background color.	Optional	
textoutstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
textoutclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
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 definition and use them multiple times by addressing them	Optional	VAR1 VAR2 VAR3 VAR4

	via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!		
Binding			
valueprop	(already explained above)		
titleprop	Property of adapter that dynamically defines the title of the control. The title is displayed as tool tip when the user moves the mouse onto the control.	Optional	
bgcolorprop	(already explained above)		
fgcolorprop	(already explained above)		
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
invisiblemode	If the visibility of the control is determined dynamically by an adapter property then there are two rendering modes if the visibility is "false": (1) "invisible": the control is not visible. (2)"cleared": the control is not visible but it still occupies space.	Optional	invisible cleared
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	

51 TOGGLE

The TOGGLE control is used to display and to edit a selection status. In principle, it acts similar to a CHECKBOX control, but it

- allows to define different icon images for the "true" and "false" representations;
- allows being informed when the user presses the CTRL or SHIFT key when clicking the icon. With this information, you can react on a combination of SHIFT and click in a different way than to a normal click or a combination of CTRL and click. This is especially useful inside grid processing when you want to allow the user to do mass selections.

Example

In the following example, the value of a boolean property is displayed by a TOGGLE control and a FIELD control.

The XML layout definition is:

```
<rowarea name="Toggle Control">
    <itr>
        <label name="Toggle" width="100" asplaintext="true">
        </label>
        <hdist width="5">
        </hdist>
        <toggle valueprop="toggleValue" flush="server"
              trueimage="images/newWithDistance.gif" falseimage="images/remove.gif"
                shiftmethod="onToggleShift" controlmethod="onToggleControl">
        </toggle>
        <hdist width="5">
        </hdist>
        <field valueprop="toggleValue" displayonly="true">
        </field>
    </itr>
</rowarea>
```

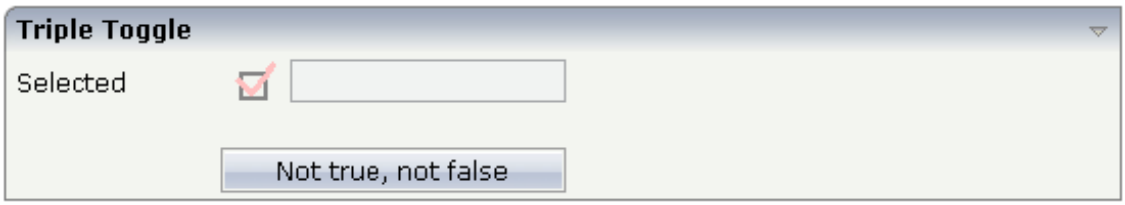
The Java adapter code is:

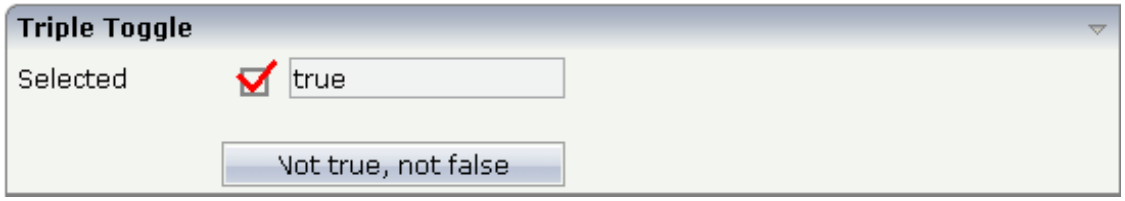
```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class CheckBoRadioButtonAdapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------

    // property >toggleValue<
    boolean m_toggleValue;
    public boolean getToggleValue() { return m_toggleValue; }
    public void setToggleValue(boolean value) { m_toggleValue = value; }

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

    public void onToggleControl()
    {
        outputMessage(MT_WARNING, "Control was pressed when clicking");
    }

    public void onToggleShift()
    {
        outputMessage(MT_WARNING, "Shift was pressed when clicking");
    }
}
```

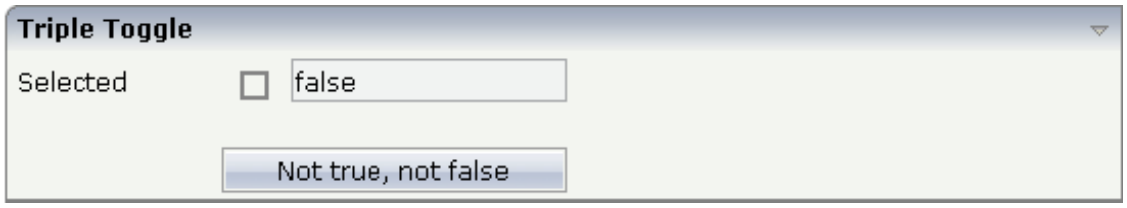
When the user presses the SHIFT or CTRL key, a corresponding method is called inside the adapter. In the example, the adapter displays the information which key was pressed in the status bar.

Usage as a Triple Status Control

Have a look at the following example:







The XML layout definition is:

```
<rowarea name="Triple Toggle">
    <itr>
        <label name="Selected" width="100" asplaintext="true">
        </label>
        <hdist width="5">
        </hdist>
        <toggle valueprop="selected_semi" flush="screen" ↩
trueimage="images/tripletrue.gif"
                falseimage="images/triplefalse.gif" shiftmethod="onToggleShift"
              controlmethod="onToggleControl" partialimage="images/triplesemi.gif">
        </toggle>
        <hdist width="5">
        </hdist>
        <field valueprop="selected_semi" displayonly="true">
        </field>
    </itr>
    <vdist height="20">
    </vdist>
    <itr>
        <hdist width="100">
        </hdist>
        <button name="Not true, not false" method="onPartialSelection_semi">
        </button>
    </itr>
</rowarea>
```

In the code, each status of the TOGGLE control is represented by a value: "true", "false" and "null".

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class CheckBoRadioButtonAdapter
    extends Adapter
{
    // property >selected<
    String m_selected="true";
    public String getSelected() { return m_selected; }
    public void setSelected(String value) { m_selected = value; }

    public void onPartialSelection()
    {
        m_selected = null;
    }
}
```

Properties

Basic			
valueprop	Name of the adapter property that represents the value of the control. Must be of type "boolean" or "Boolean". Typically the TOGGLE control knows one image to represent "true" and one image to represent "false". If using "Boolean" objects on server side (not "boolean" values) then you can also use a third image that is defined within the property PARTIALIMAGE. This image is shown if the corresponding value is "null". As consequence you can define a "triple-state-toggle" switching between "true", "false" and "null".	Obligatory	
trueimage	Image URL that is shown if the corresponding property value is "true".	Obligatory	gif jpg jpeg
falseimage	Image URL that is shown if the corresponding property value is "true".	Obligatory	gif jpg jpeg
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Appearance			
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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	
partialimage	Image URL that is shown if the corresponding property value is "null".	Optional	
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 synchronized 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

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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
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
backgroundclass	CSS style class definition that is directly passed into this control. The style class can be either one which is part of the "normal" CIS style sheet files (i.e. the ones that you maintain with the style sheet editor) - or it can be one of an other style sheet file that you may reference via the ADDSTYLESHEET property of the PAGE tag.	Optional	
Binding			
valueprop	(already explained above)		
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
shiftmethod	Name of adapter method that is invoked if the user clicks on the toggle control and presses the Shift-key the same time.	Optional	
controlmethod	Name of adapter method that is invoked if the user clicks on the toggle control and presses the Ctrl-key the same time.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour.	Optional	screen server

	Setting FLUSH to "server" means that directly after changing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.		
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	
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	

52 ACTIVEX

This is a “hot topic”: embedding ActiveX controls in pages. Before telling you what the control does, let us explain why we do it:

Of course, the client integration of ActiveX controls has some disadvantages:

- ActiveX controls are not secure: you decide to run one control or not. But do not have a “sandbox” as you have with JavaScript or with applets. Using an ActiveX control means that this control - once running - has native access to your computer, just as any other native program.
- ActiveX controls are bound to the Microsoft Windows platform.
- ActiveX controls need to be explicitly installed on the client side - maybe automated in some way, but still an explicit installation is necessary.

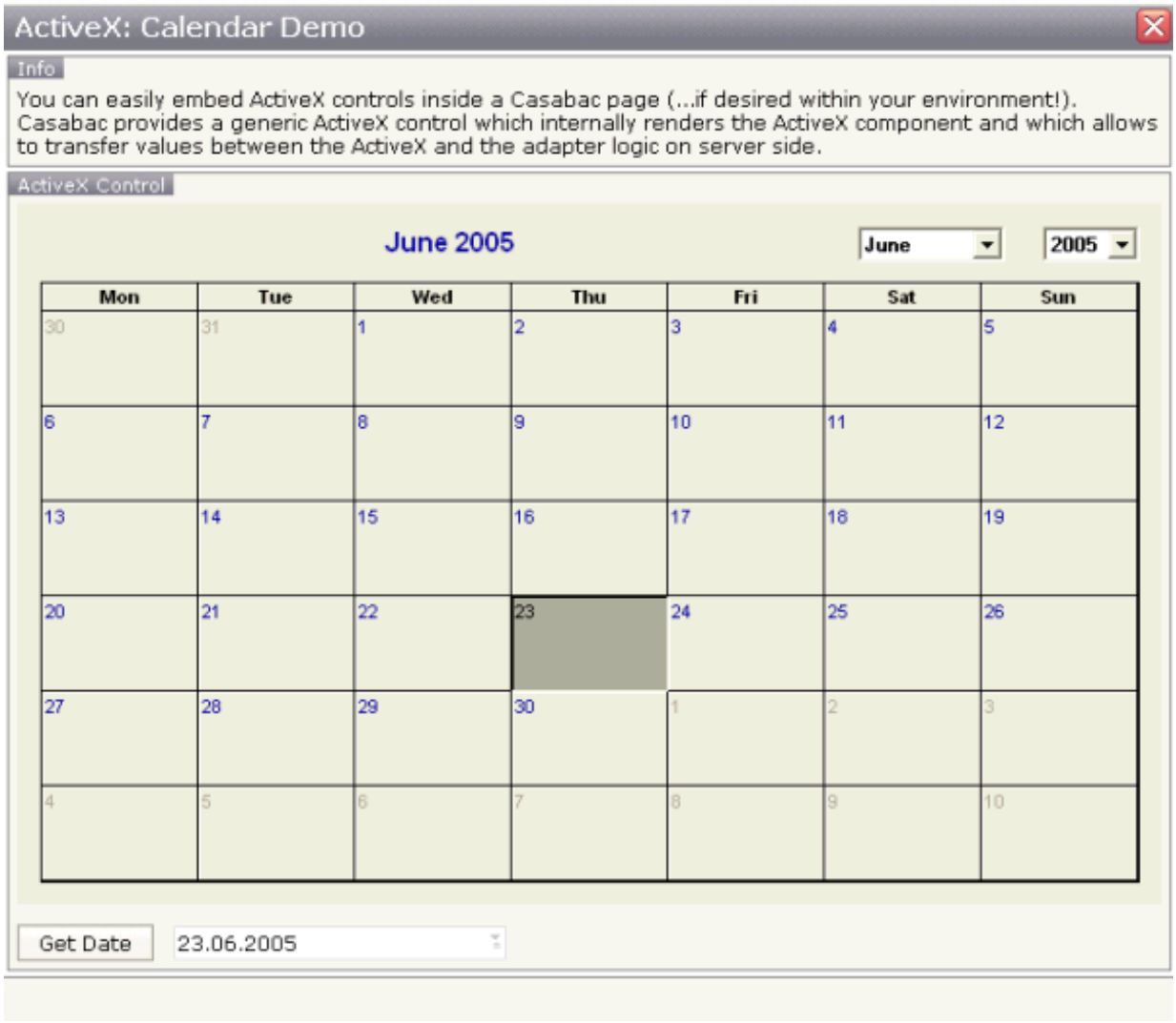
But - and this is why we support them - in some cases, they are a nice way to integrate other software which runs out of the scope of the browser.

Example: you may want to integrate your user interface with a barcode reader which is connected to your client via a serial interface. In this case, there is no way to access this barcode reader via JavaScript. You need to use an ActiveX control (or a signed applet) to connect to the serial device.

There is a simple interface between HTML/JavaScript and ActiveX, and vice versa. ActiveX controls can be embedded into an HTML page and it is possible to directly access properties of the ActiveX control from JavaScript. This interface was used for building the ACTIVEX control that you can use as an Application Designer control. Calling methods in the ACTIVEX or send/receive events is not supported.

Example

Have a look at the following screen:



Within the normal Application Designer controls, you see a calendar control: this calendar control is an ActiveX control. When choosing the **Get Date** button, the date of the calendar is displayed within the field below.

Let us have a look at the XML layout definition:

```
<rowarea name="ActiveX Control" height="100%">
    <itr takefullwidth="true" height="100%">
        <activex classid="8E27C92B-1264-101C-8A2F-040224009C02"
                 progid="MSCAL.Calendar"
                 getxparams="year;adapterYear;month;adapterMonth;day;adapterDay"
                 width="100%" height="100%">
        </activex>
    </itr>
    <vdist height="12">
    </vdist>
    <itr>
```

```
        <button name="Get Date" method="onGetDate">
        </button>
        <hdist width="12">
        </hdist>
        <field valueprop="date" width="200" datatype="date">
        </field>
    </itr>
</rowarea>
```

The ActiveX control links via a `classid` and a `progid` to the ActiveX component that is used. It has a property `getxparams`: in this property, pairs of properties are listed, each pair being the name of the ActiveX component's property and the one of the adapter property. When using `getxparams`, the ActiveX properties are transferred into the adapter properties with every roundtrip (and only when changed). There is also a `setxparams` property (not used in the example) that transfers values into the other direction: from the adapter object into the ActiveX component.

The adapter code is quite simple:

```
package com.softwareag.cis.test40;

import java.util.Date;
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.util.CDate;

public class ActiveXCalendarAdapter
    extends Adapter
{
    CDate m_date = new CDate(new Date());
    public CDate getDate() { return m_date; }
    public void setDate(CDate value) { m_date = value; }

    String m_adapterDay;
    public String getAdapterDay() { return m_adapterDay; }
    public void setAdapterDay(String value) { m_adapterDay = value; }

    String m_adapterMonth;
    public String getAdapterMonth() { return m_adapterMonth; }
    public void setAdapterMonth(String value) { m_adapterMonth = value; }

    String m_selectedDate;
    public String getSelectedDate() { return m_selectedDate; }
    public void setSelectedDate(String value) { m_selectedDate = value; }

    String m_adapterYear;
    public String getAdapterYear() { return m_adapterYear; }
    public void setAdapterYear(String value) { m_adapterYear = value; }

    public void onGetDate()
    {
        String day = m_adapterDay;
        if (day.length() == 1) day = "0" + day;
```

```
        String month = m_adapterMonth;
        if (month.length() == 1) month = "0" + month;
        m_date.setDate(m_adapterYear+month+day);
    }
}
```

The adapter's properties are automatically filled. The `onGetDate()` method assembles the properties to form a Application Designer date.

Properties

Basic			
classid	Class id of the ActiveX control. A string in the format "8E27C92B-1264-101C-8A2F-040224009C02" representing the UUID of the ActiveX component. The CLASSID is used inside the HTML client to reference the ActiveX control.	Optional	
progid	The unique program identifier which has been registered for this ActiveX Control like "Shell.Explorer"	Optional	
xinitparams	Init parameters that are used for creating an instance of the ActiveX control. Values are passed as semicolon separated string: property;value;property;value etc. The property is the name of the ActiveX control's property that is initialized with the corresponding value.	Optional	
setxparams	Same as GETXPARAMS but now the other direction. Adapter properties that are transferred (on change) into corresponding ActiveX properties with each response. The string format is the same: activeXProperty;adapterProperty;activeXProperty;adapterProperty etc.	Optional	
getxparams	Semicolon separated list of which ActiveX control are linked with which adapter properties. The format is: activeXProperty;adapterProperty;activeXProperty;adapterProperty etc. With each request send from the browser the ActiveX properties are collected in from the ActiveX control and are transferred (if they have changed) into the corresponding adapter properties.activex_attr_progid"Program id of the ActiveX control. E.g. "MSCAL.Calendar" for the Microsoft calendar.	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").	Optional	100 120 140 160 180

	(C) Percentage sizing: input a percentage 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.		200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
reloadprop	Indicates that the ActiveX component is reloaded with every response from the server that changed data of the ActiveX component.	Optional	
useparamtag	Set to false if setting the parameters in your ActiveX does not work using the html param tag. Normally you don't have to set this attribute.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

53 GOOGLEMAP2

The GOOGLEMAP2 control is used to provide for Google Maps support within Application Designer pages. The control internally makes use of the Google Maps API. In order to use the control on your site, you need to sign up for a Google Maps API key at ***https://developers.google.com/maps/signup***. Make sure that you agree with the Google Maps APIs Terms of Service (***https://developers.google.com/maps/terms***).

Before You Start

In order to use the GOOGLEMAP2 control, you need to sign up for a Google Maps API key. A key is valid for a single "directory" on your web server only, i.e. you sign up for a URL like *http://www.mysite.com/mywebapp/myproject*. With a standard installation of Application Designer on localhost, you may sign up for the URL *http://localhost:8080/mywebapp/myproject*. Typically, you develop your Application Designer web application not on the site on which you run it later in productive mode. Therefore, you may sign up for two different sites (development and production site).

Required Steps

1. Choose the project directory that keeps the layouts using the GOOGLEMAP2 control.
2. Sign up for a Google Maps API key at ***https://developers.google.com/maps/signup*** for this project directory (e.g. *http://localhost:8080/mywebapp/myproject*).
3. Create the API key page. Store the key page in the registered project directory. You are free in naming the file (the file extension must be "html"). The GOOGLEMAP2 control embeds your API key as a subpage. The subpage must have the following minimum structure:

```
<html>
  <head>
    <script src=" ↩
http://maps.google.com/maps?file=api&amp;v=2.x&amp;key=YOUR_API_KEY"></script>
    <script src="../HTMLBasedGUI/general/googlemapsscript.js"></script>
  </head>
  <body>
    <div id="map" style="position:absolute; top0; left:0;"></div>
  </body>
</html>
```

You see that the page includes two JavaScript libraries. The first line refers to the Google Maps API. Replace the placeholder "YOUR_API_KEY" with your Google Maps API key. With the second line, the page includes the control's scripting (calls from Application Designer to the Google Maps). The page body is quite simple: it contains a single `div` tag with the ID "map". This `div` is used as an anchor to insert Google Maps controls dynamically.

Example

General Usage

The map options are taken from the property `infoprop`. On this object, you may set the address (or latitude and longitude), the zoom level and the map size as well as the map type.

Note: The usage of address or longitude/latitude is mutually exclusive.

The above map is controlled by the following adapter code:

```
public class GoogleMap2Adapter extends Adapter
{
  // property >gm2Info<
  GOOGLEMAP2Info m_gm2Info = new GOOGLEMAP2Info(
                               GOOGLEMAP2Info.NO_MAPTYPE_CONTROL,
                               GOOGLEMAP2Info.SMALL_MAP);
  public GOOGLEMAP2Info getGm2Info(){ return m_gm2Info; }
  public void setGm2Info(GOOGLEMAP2Info value){ m_gm2Info = value; }
  // Marker items
  private class HotelMarker extends GOOGLEMAP2Item
  {
    // [see section "Marker Management"]
    ....
```

```
}

private Hashtable hotels = new Hashtable();

/** initialisation - called when creating this instance */
public void init()
{
  m_gm2Info.setAddress("Darmstadt, Germany");
  m_gm2Info.setZoomlevel("13");
  setupHotels();
}

// property >hotelSelection<
String m_hotelSelection = "";
public String getHotelSelection(){ return m_hotelSelection; }
public void setHotelSelection(String value){ m_hotelSelection = value; }

// property >validHotSel<
COMBODYNValidValues m_validHotSel = new COMBODYNValidValues();
public COMBODYNValidValues getValidHotSel(){ return m_validHotSel; }

/** */
public void onSelect()
{
  HotelMarker hotel = (HotelMarker) hotels.get(m_hotelSelection);
  m_gm2Info.centerMarker(hotel);
}

// property >hotelDesc<
String m_hotelDesc = "";
public String getHotelDesc(){ return m_hotelDesc; }
public void setHotelDesc(String value){ m_hotelDesc = value; }

// property >hotelName<
String m_hotelName = "";
public String getHotelName(){ return m_hotelName; }
public void setHotelName(String value) { m_hotelName = value; }

/** */
public void onPlaceOwn()
{
  if (m_hotelName.equals(""))
  {
    outputMessage(MT_ERROR, "Please specify a name.");
    return;
  }

  HotelMarker MyHotel = new HotelMarker(m_hotelName);
  MyHotel.setInfoText("<b>" + m_hotelName + "</b>\n" + m_hotelDesc);

  m_gm2Info.addMarkerToLastSelectedPoint(MyHotel);
}
```

```
/** */
public void onRemove()
{
  m_gm2Info.removeLastSelectedMarker();
}

/** */
public void onRemoveAll()
{
  m_gm2Info.clear();
  hotels.clear();
  m_validHotSel.clear();
  m_hotelSelection = "";
}

/** */
public void onShowAll()
{
  onRemoveAll();
  setupHotels();
}

private void setupHotels()
{
  setupHotel("Bestwestern, Parkhaus-Hotel",
      "Grafenstraße 31, 64283 Darmstadt");
  setupHotel(" ....
  // deactivate last added marker
  m_gm2Info.setSelectedMarker(null);
}

private void setupHotel(String name, String address)
{
  HotelMarker hotel = new HotelMarker(name, address);
  hotel.setInfoText("<b>" + name + "</b>\n" + address.replaceAll(", ", "\n"));
  m_gm2Info.addMarker(hotel, false);
  if (name.length() > 23)
    name = name.substring(0, 23) + "...";
  m_validHotSel.addValidValue(String.valueOf(hotel.getId()), name);
  hotels.put(String.valueOf(String.valueOf(hotel.getId())), hotel);
}

// property >naviCity<
String m_naviCity;
public String getNaviCity(){ return m_naviCity; }
public void setNaviCity(String value){ m_naviCity = value; }

// property >naviCountry<
String m_naviCountry;
public String getNaviCountry(){ return m_naviCountry; }
public void setNaviCountry(String value){ m_naviCountry = value; }
```

```
  // property >naviStreet<
  String m_naviStreet;
  public String getNaviStreet(){ return m_naviStreet; }
  public void setNaviStreet(String value){ m_naviStreet = value; }

  /** */
  public void onNavigate()
  {
    String address = "";

    if (!m_naviStreet.equals(""))
    {
      address += m_naviStreet + ", ";
    }
    if (!m_naviCity.equals(""))
    {
      address += m_naviCity + ", ";
    }
    if (!m_naviCountry.equals(""))
    {
      address += m_naviCountry;
    }

    m_gm2Info.setAddress(address);
  }

  ....
}
```

The above map is initialized with the instantiation of the `GOOGLEMAP2Info` object and just a few simple lines of code in the `init()` method.

The constructor of the `GOOGLEMAP2Info` class takes the following arguments:

- **Map Type Control Setting**
  Using the constant "MAPTYPE_CONTROL" (instead of "NO_MAPTYPE_CONTROL" which is used in the above example) would result in three buttons in the upper right corner of the map, which enable the user to change the map view between "Map", "Satellite" and "Hybrid" mode.

  Note: The range of zoom levels may differ for different map types in the same region.

- **Map Size Setting**
  For the above map the map size property is set to the constant "SMALL_MAP" which results in the four navigation arrows and the zoom buttons in the upper left corner. The constant "LARGE_MAP" would alternatively provide more precise navigation controls allocating more of the map area in exchange.

The GOOGLEMAP2Info class provides for a second constructor without any arguments. Using this constructor is equal to the usage of the described constructor with the constants "NO_MAP-TYPE_CONTROL" and "SMALL_MAP".

In the init() method, the map view is positioned via the setAddress method. The same result would be achieved using the setLatLng method with the argument "49,879046" (for latitude) and "8,670112" (for longitude). It is obligatory to set the map view using one of these variants or using a marker (see *Marker Management* for further information). Otherwise the map will not be displayed.

The range of values for the zoomlevel property may vary according to the map region. The value "4" is used by default if zoomlevel is not set explicitly.

The GOOGLEMAP2 control listens to changes on the address (or latitude/longitude) and the zoomlevel property.

Marker Management

To use the marker management of the GOOGLEMAP2 control, you need an implementation of the GOOGLEMAP2Item class. For the above example, the following code was used:

```
private class HotelMarker extends GOOGLEMAP2Item
{
  private String m_name;

  public HotelMarker(String name)
  {
    super(true);
    m_name = name;
  }

  public HotelMarker(String name, String address)
  {
    super(address, true);
    m_name = name;
  }

  public void reactOnSelect()
  {
    outputMessage(MT_SUCCESS, "Hotel '" + m_name + "' selected.");
  }

  public void reactOnDrag()
  {
    outputMessage(MT_SUCCESS, "Hotel '" + m_name + "' has moved.");
  }

  public void reactOnDeactivate()
  {
    outputMessage(MT_SUCCESS, "Hotel '" + m_name + "' deselected.");
```

```
    }
}
```

The methods `reactOnSelect()`, `reactOnDrag()` and `reactOnDeactivate()` have to be implemented in order to define the behavior for the following events:

- **Select**
  The user clicks on the corresponding marker on the map.

  If the user clicks the button, for example, five times, this event is fired five times, even if the button remains active.
- **Drag**
  The user drags the marker to a different position on the map and drops it.

  It is possible to switch dragging on and off for each marker using the marker's `setDraggable(boolean)` method. By default, all markers are draggable.
- **Deactivate**
  The user clicks a different marker or somewhere else on the map when the marker is active.

  A marker is considered active from selection until deactivation.

Markers may be added to specific positions or to the position the user has clicked last on the map, removed, activated, deactivated or centered using the `infoprop` property. As mentioned above in the section ***General Usage***, a marker may be used to set the map's view if it is told to center on the marker.

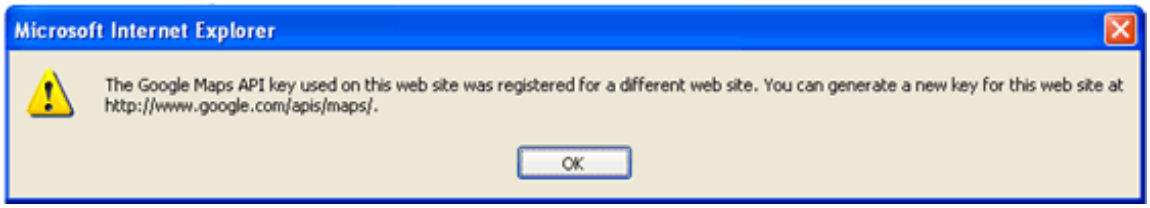
Each marker may have an `infoText` that is shown within the pop-up when the marker is selected by the user. Changes to this text will be updated on the client side. If no text is set, a pop-up will not appear. Since the `infoText` is treated as HTML code, it may be formatted like HTML. Only breaks will automatically be replaced.

The GOOGLEMAP2 control listens to changes on markers, address (or latitude/longitude) and `infoText` property.

Typical Problems

Google Map API Key

Your Google Maps API key is bound to a directory on a certain web server (i.e. you sign up for the URL *http://mycomputer.mydomain.com:8080/mywebapp/myproject*). If you use your key for another URL, Google shows an error message:



Reasons that cause the error:

- You have registered your computer using the computer's name (e.g. *http://mycomputer...*). But the Application Designer development workplace is started using the URL *http://localhost....*

  Solution: start the Application Designer workplace with *http://mycomputer....*
- The registered directory (e.g. *.../mywebapp/myproject*) does not match your installation (either a mistake in writing when signing up for the key or you have renamed the web application or project after registration).

  Solution: rename your web application or project to match the registered names. Or sign up for a new key and insert the new key into the API key page. In the latter case, delete the content of the browser's cache. Otherwise, the browser will use the former API key page (and thus the old key).

Map Remains Gray

If you use longitude and latitude for placing the marker on the map, their values may exceed the map top (or bottom) border. If you are able to find the map by scrolling down (or up), then this is the case. Check the values for longitude and latitude in this case.

Properties

Basic			
infoprop	Name of adapter property representing the control on server side. The property must be of type GOOGLEMAPInfo. Read further information inside the Java API Documentation.	Obligatory	
apikeypagename	Name of the Maps API Key page. Example: mygooglemapsapikey.html. Keep this file within the project directory (directory within the CIS HTML pages are kept). The GOOGLEMAP-control expects this file within certain Javascript includes and content. Have look into chapter "Google Map - Before You Start" within the Developers Guide	Obligatory	
width	Width of the control. There are three possibilities to define the width:	Optional	100 120

	(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 percentage 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.		140 160 180 200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	

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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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 synchronized 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

54 HELPICON

For detailed information on the HELPICON control, see *Online Help Management*.

55 REPORT

For detailed information on the REPORT control, see *Writing Reports* in the *Special Development Topics*.

56 ROWCHARTAREA

The ROWCHARTAREA control allows you to arrange and visualize objects.

Example

In this example, we will use the ROWCHARTAREA control to develop a very simple “FlowChart Modeller” step-by-step.

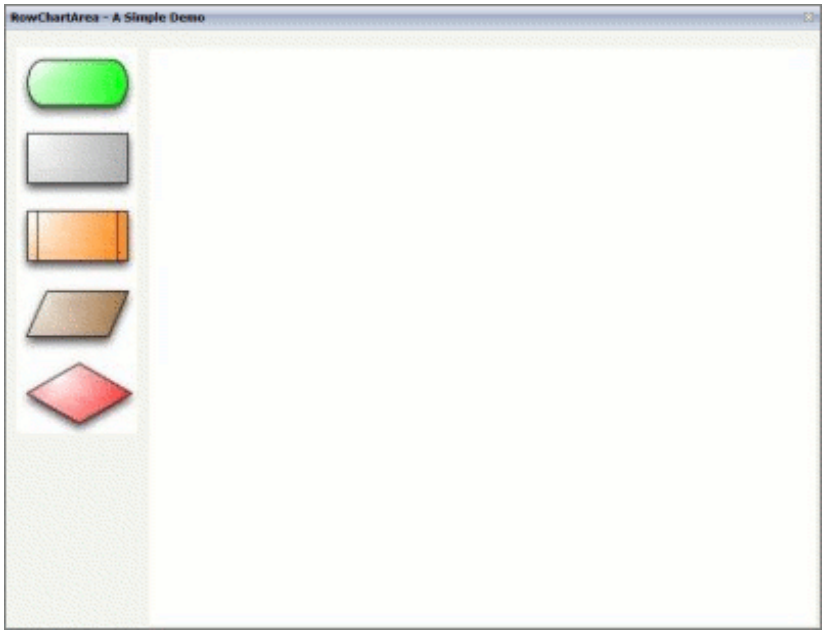
This is a large example with lots of Java code. However, you will notice that a large amount of the code is not unknown to you. Detailed explanations are provided in the sample code. Therefore, you should also take close a look at the comment sections in the sample code.

This example consists of the steps below.

- Step 1 - Creating a Simple ROWCHARTAREA with Icons
- Step 2 - Adding Labels to the Items
- Step 3 - Drawing Connection Lines
- Step 4 - Showing and Hiding the Connection Spots
- Step 5 - Adding Context Menus

Step 1 - Creating a Simple ROWCHARTAREA with Icons

We will first design the following layout which contains a ROWCHARTAREA control in which you can drop and arrange different icons.



The XML layout definition is:

```
<pagebody takefullheight="true">
  <itr height="100%">
    <coltable0 width="110" takefullheight="true">
      <rowarea height="100%">
        <vdist></vdist>
        <itr>
          <dropicon image="../cisdemos/images/flowchart_01.jpg"
               draginfo="flow01">
          </dropicon>
        </itr>
        <itr>
          <dropicon image="../cisdemos/images/flowchart_02.jpg"
               draginfo="flow02">
          </dropicon>
        </itr>
    ...
    // Some more 'Drop Icons'. Define them in the same way as above
    ...
        <vdist height="100%"></vdist>
      </rowarea>
    </coltable0>
    <coltable0 width="100%" takefullheight="true">
      <rowarea height="100%">
        <rowchartarea infoprop="chartareaprop" width="100%" height="100%"
         usegridlines="true" gridlinexdistance="50" gridlineydistance="50">
        </rowchartarea>
      </rowarea>
    </coltable0>
  </itr>
</pagebody>
```

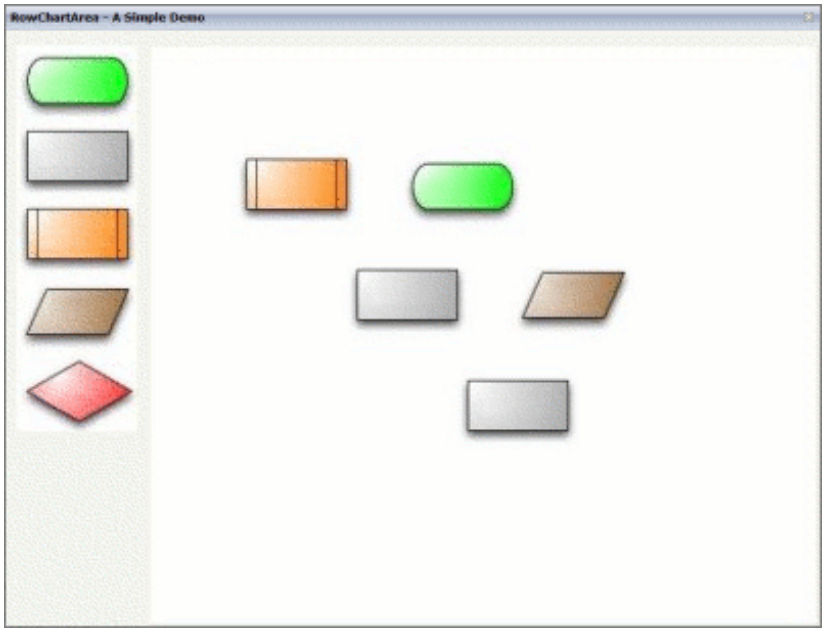
The Java code of the adapter is:

```
public class RowChartAreaSimpleDemoAdapter
extends Adapter
{
  /**
   * Name of the adapter property
   * that represents the control on server side.
   * */
  public class RowChartArea extends CHARTAREAInfo
  {
    public RowChartArea() {}

    /**
     * This overwritten method is needed.
     * It takes control of what happens with
     * the dropped objects.
     * */
    public void reactOnDrop(String dropInfo, int xpos, int ypos)
    {
        // On drop, a new FlowChartItem is generated
```

```
      FlowChartItem fci = drawFlowChartItem(dropInfo, xpos, ypos);
      // and added to this RowChartArea property
      addChartAreaItem(fci);
  }
}
/**
 * This class represents the dropped and visualized items
 * inside the ROWCHARTAREA control.
 *  */
public class FlowChartItem extends CHARTAREAItem
{
  // ID for the item type (flowchart_01, flowchart_02, ...).
  int m_itemId;

  // Property for labeling the items.
  CHARTAREAItemText m_txtout = null;
  public CHARTAREAItemText getTxtout() { return m_txtout; }
  public void setTxtout(CHARTAREAItemText txtout)
  {  this.m_txtout = txtout;}

  // This rectangle is needed to show which item is selected
  CHARTAREAItemRect m_rect = null;
  public CHARTAREAItemRect getRect() { return m_rect; }
  public void setRect(CHARTAREAItemRect rect) { this.m_rect = rect; }

  public FlowChartItem(int x, int y, int width,
     int height, String dragIcon, int itemId)
  {
      super(x, y, width, height, dragIcon);
      m_itemId = itemId;
  }

  /**
   * reactOnSelection() and reactOnContextMenuRequest()
   * have to be implemented
   * We will have a closer look at these methods later.
   * */

  /** take care of ContextMenu for the FlowChartItems*/
  public void reactOnContextMenuRequest()
  {  }

  /** take care of what happens if this item is selected  */
  public void reactOnSelection(boolean shiftKey, boolean ctrlKey)
  {
    // save the info which item is last selected
    m_lastSelected = this;
  }
}

/**
 * generate a FlowChartItem
```

```
 * */
private FlowChartItem drawFlowChartItem(String dropInfo, int xpos,
        int ypos)
{
  FlowChartItem fci = null;
  String img = null;

  int itemID = new Integer(dropInfo.substring(dropInfo.length()-1,
                  dropInfo.length())).intValue();

  switch(itemID)
  {
  case 1:
      img = "images/flowchart_01.jpg";
      break;
  case 2:
      img = "images/flowchart_02.jpg";
      break;
  ...
  // some more icons if needed
  ...
  }

  // create a new FlowChartItem
  fci = new FlowChartItem(xpos, ypos, 120, 80,"../cisdemos/"+img,
           itemID);

  // if the object was successfully created, add an image
  if (fci != null)
    fci.addImage(5,5, 1,110,70, img);

  return fci;
  }

  /** Members */
  CHARTAREAInfo m_chartareaprop = new RowChartArea();
  public CHARTAREAInfo getChartareaprop() { return m_chartareaprop; }
  public void setChartareaprop(CHARTAREAInfo value)
   { m_chartareaprop = value; }

  // property that stores the information which item is the last selected
  FlowChartItem m_lastSelected = null;

  /** initialisation - called when creating this instance*/
  public void init()
  { }
```

Let us have a look at what we have done so far. In the following example, you can see our layout with some dropped icons.



Until now, this example only allows you to drag-and-drop the `FlowChartItem` icons into the ROWCHARTAREA control where you can move the icons.

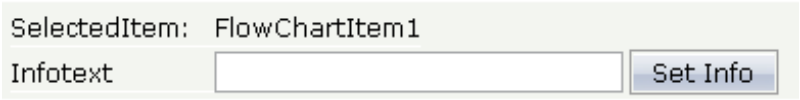
Step 2 - Adding Labels to the Items

We will now add a label to each `FlowChartItem`, and we will add a visualization for the item which is currently selected. Therefore, we need to add some more XML and Java code.

First, we will add the following to the XML layout definition:

```
<itr align="left">
  <label name="Selected Item:" width="100" asplaintext="true"></label>
  <field valueprop="selectedItem" width="200" displayonly="true"
     noborder="true"></field>
</itr>
<vdist></vdist>
<itr align="left">
  <label name="Infotext:" width="100" invisiblemode="cleared"
     asplaintext="true"></label>
  <field valueprop="infotext" width="200" statusprop="fstatus"
     maxlength="10" noborder="true"></field>
  <hdist></hdist>
  <button name="Set Info" method="onSetInfo" width="75"></button>
</itr>
```

The new lines look as follows:



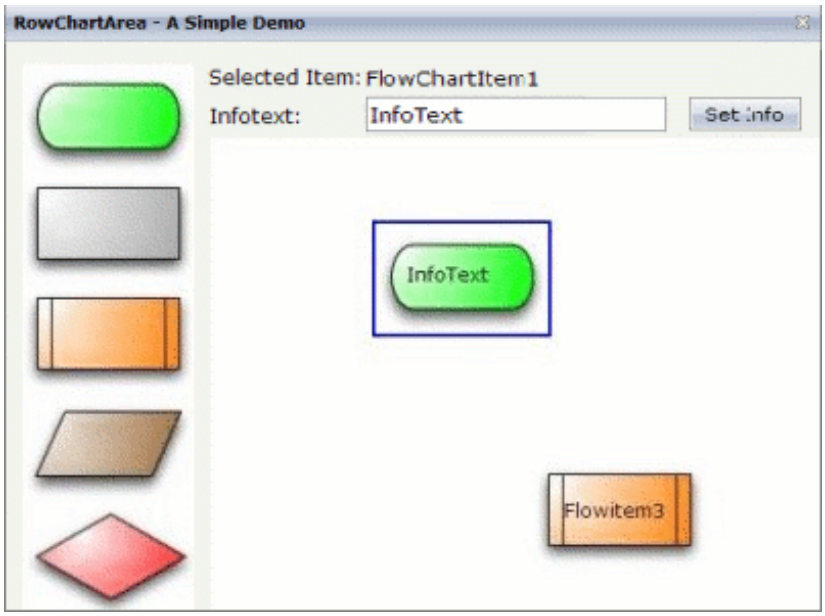
The Java code of the adapter is (new code is indicated in bold):

```
public class FlowChartItem extends CHARTAREAItem
  {
    ...
    /**
     * take care of what happens if this item is selected
     *  */
    public void reactOnSelection(boolean shiftKey, boolean ctrlKey)
    {
  // ==> new
      // bring the infotext of this item to the screen
      m_infotext = m_txtout.getText();
      // and show the itemId
      m_selectedItem = "FlowChartItem "+m_itemId;
      // set the m_infotext field status to edit
      m_fstatus = CS_EDIT;
     // call prvate method setSelected()
      setSelected(this);
  // <== new
    }
  }
...
  private FlowChartItem drawFlowChartItem(String dropInfo, int xpos,
          int ypos)
  {
  ...
    // if the object was successfully created add an image
    if (fci != null)
    {
      fci.addImage(5,5, 1,110,70, img);
  // ==> new
      // create and add a rectangle to the item
      // this is for the visualization of the selected item
      CHARTAREAItemRect rect = fci.addRect(3, 3, -1, 114 , 74, 0,"", "");
      fci.setRect(rect);

      // create and add an empty textfield to the item
      CHARTAREAItemText outputText =
                 fci.addText(25, 30, 2, 85, 15, "","","");
      fci.setTxtout(outputText);
  // <== new
    return fci;
  }

/** Members */
...
```

```
  // ==> new
    // property >infotext<
    String m_infotext;
    public String getInfotext() { return m_infotext; }
    public void setInfotext(String value) { m_infotext = value; }

    // property >fstatus<
    String m_fstatus= CS_DISPLAY;
    public String getFstatus() { return m_fstatus; }
    public void setFstatus(String value) { m_fstatus = value; }

    // property >selectedItem<
    String m_selectedItem;
    public String getSelectedItem() { return m_selectedItem; }
    public void setSelectedItem(String value) { m_selectedItem = value; }
  // <== new

...

  // ==> new
  /**
   * Sets the color of the rectangle around the selected item
   * to #0000FF and stores the information which item is the
   * last selected one in m_lastSelected.
   * setSelected(null) resets the color of the selected item to ""
   **/
  private void setSelected(FlowChartItem item)
  {
    if(m_lastSelected != null && m_lastSelected != item)
      m_lastSelected.getRect().setBackgroundColor("");
    if(item != null)
    {
      item.getRect().setBackgroundColor("#0000FF");
      m_lastSelected = item;
    }
  }
  // <== new

...

  // ==> new
  /**
   *  public methods that sets the infotext
   *  and the tooltip to the last selected item.
   *  The method is called if the button 'Set Info' is clicked.
   * */
  public void onSetInfo()
  {
    if(m_lastSelected == null)
      return;
    if(!m_infotext.equalsIgnoreCase(m_lastSelected.getTxtout().getText())&&
        m_infotext != null )
```

```
    {
      // set the infotext of the item
      m_lastSelected.getTxtout().setText(m_infotext);
      // and the tooltip
      m_lastSelected.setTooltip(m_infotext);
    }
  }
  // <== new
...
```

Now, we have a label for each `FlowChartItem` and a visualization for the selected item.



Step 3 - Drawing Connection Lines

We will now draw lines between the single `FlowChartItem` icons.

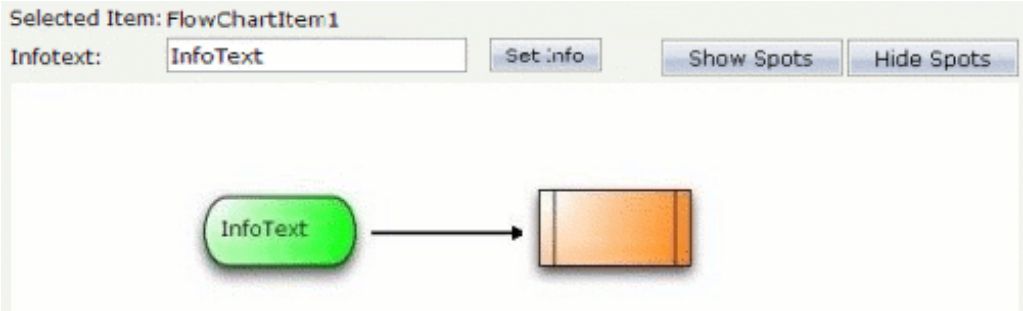
For this purpose, changes to the XML layout are not required. However, we have to add some more Java code:

```
  // ==> new
  /**
   * a class for the small spots where the connection lines
   * are drawn from/to
   *  */
  public class ConnectionSpot extends CHARTAREASpot
  {
    public ConnectionSpot(int x, int y, int z, int orientation)
    {
      super(x, y, z, orientation);
    }
  /** This method has to be overwritten.
   *  It generates the lines between the items.
   */
    public CHARTAREAConnectionLine buildNewConnectionLine
          (CHARTAREASpot fromSpot, CHARTAREASpot toSpot)
    {
       //Create a new line
      CHARTAREAConnectionLine line = new
           CHARTAREAConnectionLine(fromSpot,toSpot);
      // set line style
      line.setToSpotStyle(CHARTAREAConnectionLine.LINE_STYLE_ARROW);
      line.setLineColor("000000");
      line.setZIndex(-1);
      return line;
    }
  }
  // <== new
...
  private FlowChartItem drawFlowChartItem(String dropInfo, int xpos,
          int ypos)
  {
  ...
    // if the object was successfully created ...
    if (fci != null)
    {
    ...
  // ==> new
    /**
     *  add the connection spots
     *  each items gets 4
     *  north, south, east and west
     *  */
      fci.addHotspot(new ConnectionSpot(55,0,3,CHARTAREASpot.O_NORTH));
      fci.addHotspot(new ConnectionSpot(55,70,3,CHARTAREASpot.O_SOUTH));
      fci.addHotspot(new ConnectionSpot(110,35,3,CHARTAREASpot.O_EAST));
      fci.addHotspot(new ConnectionSpot(0,35,3,CHARTAREASpot.O_WEST));
  // <== new
}
```

Now, each `FlowChartItem` has connection spots and it is thus possible to connect the items with a line.

Step 4 - Showing and Hiding the Connection Spots

We will now add buttons for showing and hiding the connection spots to the layout.



This is the corresponding XML layout definition:

```
<hdist width="18"></hdist>
<button name="Hide Spots" method="onHideSpots" width="100"></button>
<hdist></hdist>
<button name="Show Spots" method="onShowSpots" width="100"></button>
```

The Java code of the adapter is:

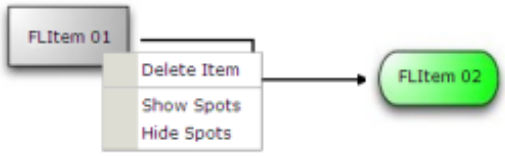
```
/** Members */
...
  // ==> new
// property that stores the information whether the spots are (in)visible private ↩
boolean m_hotSpotVisible = true;
  // <== new

...

  // ==> new
  /** Makes the 'Connection Spots' invisible */
  public void onHideSpots()
  {
    // save the information whether the spots are visible ore not
    m_hotSpotVisible = false;
    // make the spots invisible
    m_chartareaprop.setAllHotspotsVisible(m_hotSpotVisible);
    // no item selected => no rectangle visible
    setSelected(null);
  }
  /** Makes the 'Connection Spots' visible */
  public void onShowSpots()
  {
    // save the information whether the spots are visible ore not
    m_hotSpotVisible = true;
    // Make the spots visible
    m_chartareaprop.setAllHotspotsVisible(m_hotSpotVisible);
```

```
  }
  // <== new
```

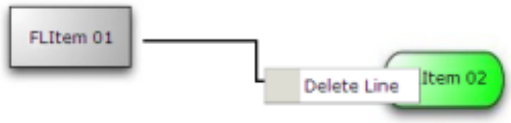
Step 5 - Adding Context Menus

The will now add three different content menus, where each context menu has its own functionality:

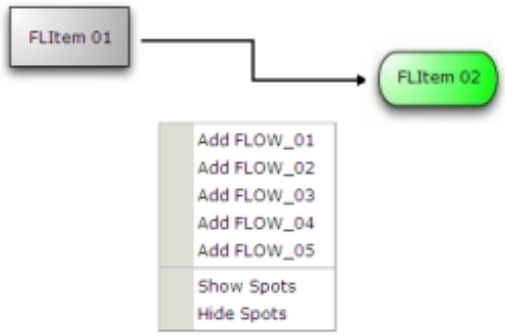
- A context menu for the `FlowChartItem` icons:



- A context menu for the lines:



- A context menu for the ROWCHARTAREA:



The Java code for the context menus is:

```
/** Members */
  // ==> new
  FlowChartItem m_itemWithContext = null;
  CHARTAREAConnectionLine m_lineWithContext = null;

  // three properties for the different contextmenus
  TREECollection m_contextMenuLine = new TREECollection();
  TREECollection m_contextMenuItem = new TREECollection();
  TREECollection m_contextMenuArea = new TREECollection();
```

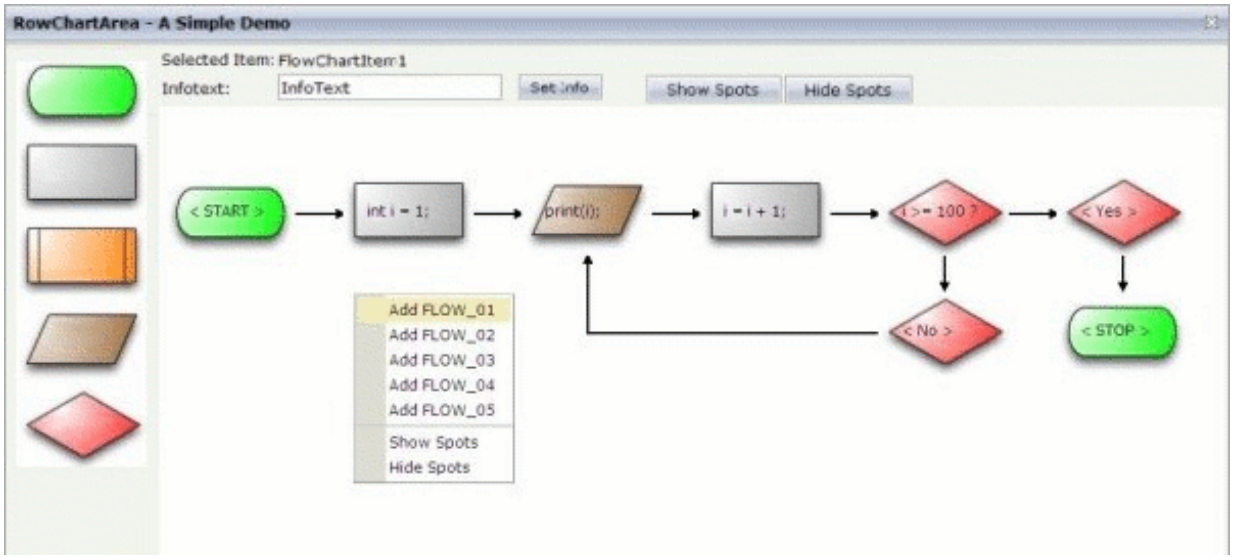
```
  /** initialisation - called when creating this instance*/
  public void init()
  {
    // build the contextmenus in the init method
    buildContextMenus();
  }
  // <== new

...

  // ==> new
    /** build the contextmenus */
    private void buildContextMenus()
    {
      LineContextMenuItem node;

      // contextmenu items for the area
      node = new LineContextMenuItem("Add FLOW_01", 1);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Add FLOW_02", 2);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Add FLOW_03", 3);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Add FLOW_04", 4);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Add FLOW_05", 5);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem(MENUNODEInfo.TYPE_SEPARATOR);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Show Spots", 12);
      m_contextMenuArea.addTopNode(node, true);
      node = new LineContextMenuItem("Hide Spots", 13);
      m_contextMenuArea.addTopNode(node, true);

      // contextmenu items for the FlowChartItems
      node = new LineContextMenuItem("Delete Item", 11);
      m_contextMenuItem.addTopNode(node, true);
      node = new LineContextMenuItem(MENUNODEInfo.TYPE_SEPARATOR);
      m_contextMenuItem.addTopNode(node, true);
      node = new LineContextMenuItem("Show Spots", 12);
      m_contextMenuItem.addTopNode(node, true);
      node = new LineContextMenuItem("Hide Spots", 13);
      m_contextMenuItem.addTopNode(node, true);

      // contextmenu for the lines
      node = new LineContextMenuItem("Delete Line", 21);
      m_contextMenuLine.addTopNode(node, true);
    }
    /**
     * The class for the contextmenu items
     *  */
```

```
  public class LineContextMenuItem extends MENUNODEInfo
  {
    /** keeps the information what action should be called in the
     * reactOnContextMenuSelect() method... */
    int m_action;

    public LineContextMenuItem(String text)
    {
      super(text);
      m_action = -1;
    }

    public LineContextMenuItem(String text, int action)
    {
      super(text);
       m_action = action;
    }

    public LineContextMenuItem(String text, String image, int action)
    {
      super(text, image);
      m_action = action;
    }

    public void reactOnSelect()
    {
        reactOnContextMenuSelect(m_action);
    }
}

/**
 * React on a click in the contextmenu.
 * depending on the m_action content of the clicked item
 * there are different 'reactions'.
 */
public void reactOnContextMenuSelect(int action)
{
  switch(action)
  {
    // new FlowChartItem type 01 to added
    case 1:
      m_chartareaprop.reactOnDrop("flow01",
        m_chartareaprop.getMouseDownX(),m_chartareaprop.getMouseDownY());
    break;
    // new FlowChartItem type 02 to added
    case 2:
      m_chartareaprop.reactOnDrop("flow02",
        m_chartareaprop.getMouseDownX(),m_chartareaprop.getMouseDownY());
    break;
    ...
    // one reaction for each possible ‘FlowChartItem’
    ...
```

```
      // delete the selected 'FlowChartItem
      case 11:
      m_itemWithContext.removeAllConnections();
      m_chartareaprop.removeChartAreaItem(m_itemWithContext);
      m_itemWithContext = null;
      setSelected(null);
      m_selectedItem = "";
      m_infotext = "";
      m_fstatus = "DISPLAY";
      break;
      // Make 'Connection Spots' visible
      case 12:
        onShowSpots();
      break;
      // Make 'Connection Spots' invisible
      case 13:
        onHideSpots();
      break;
      // delete selected line
      case 21:
        m_lineWithContext.removeLine();
        m_lineWithContext = null;
      break;
    }
  }
  // <== new
...
  public class RowChartArea extends CHARTAREAInfo
  {
...
    /** reactOnContextMenu for area */
    public void reactOnContextMenuRequest()
    {
  // ==> new
    // open the context menu for the area
      showPopupMenu(m_contextMenuArea);
  // <== new
    }
  // ==> new
    /** reactOnContextMenu for line */
    public void
     reactOnContextMenuRequestConnectionLine(CHARTAREAConnectionLine
       connectionLine)
    {
      m_lineWithContext = connectionLine;
    // open the context menu for the line
      showPopupMenu(m_contextMenuLine);
    }
  // <== new
  }
...
  public class FlowChartItem extends CHARTAREAItem
```

```
{
  ...

// ==> new
  /** reactOnContextMenu for Item*/
  public void reactOnContextMenuRequest()
  {
    m_itemWithContext=this;
    showPopupMenu(m_contextMenuItem);
  }
// <== new

  ...
}
```

The result of our coding is a simple “for-loop” flowchart. For example (with an opened context menu):



Properties

Basic			
infoprop	Name of the adapter property that represents the control on server side. Return type must be "CHARTAREAInfo".	Obligatory	
width	Width of the control. There are three possibilities to define the width:	Optional	100 120

	(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 percentage 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.		140 160 180 200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
pagebaseddragdrop	Default is false. Allows to 'drag and drop' icons from another frame into the ROWCHARTAREA control. Set to true if you want to enable 'drag and drop' only within the page the ROWCHARTAREA control is placed in.	Optional	true false
usegridlines	Enables/Disables 'automatic snap to grid'. Default is true. You can define the X- and Y-Distance of the grid.	Optional	true false
gridlinesprop	Name of an adapter property that provides the information if 'automatic snap to grid' is enabled or disabled. As consequence you can control the 'automatic snap to grid' function of the control dynamically.	Optional	

	The server side property needs to be of type "boolean".		
gridlinexdistance	Define the X-Distance of the gridlines. Default is 50. Notice: The X-Distance does not have any effect if 'automatic snap to grid' is disabled. Please have a look at the USEGRIDLINES and GRIDLINESPROP which are already explained above.	Optional	1 2 3 int-value
gridlineydistance	Define the Y-Distance of the gridlines. Default is 50. Notice: The Y-Distance does not have any effect if 'automatic snap to grid' is disabled. Please have a look at the USEGRIDLINES and GRIDLINESPROP which are already explained above.	Optional	1 2 3 int-value

57 TIMER

For detailed information on the TIMER control, see *Non-Visual Controls and Hot Keys* in the *Special Development Topics*.

IV Working with Grids

So far, all pages that were explained referred to singular properties of the adapter class. There was just one control referring to one property (and sometimes more than one).

This part shows you how to deal with grids. Working with grids is as simple as working with singular properties because the grid management adapts seamlessly into the normal processing of the Application Designer environment.

The information provided in this part is organized under the following headings:

58 Basics

It is quite simple: “normal” controls refer to an adapter class and are bound to singular properties - i.e. properties that are accessible by a simple `set`/`get` method. Grid controls refer to an adapter class as well - but are bound to a collection of objects. Each collection element provides `set`/`get` methods to access its content, i.e. its own properties.

Two types of grid controls are available:

- The TEXTGRID2 control is a control that displays grid data - but does not allow any change to the data. You can select grid rows and colorize them in different ways. Change the order of columns dynamically and sort columns by clicking into the title row of the grid.

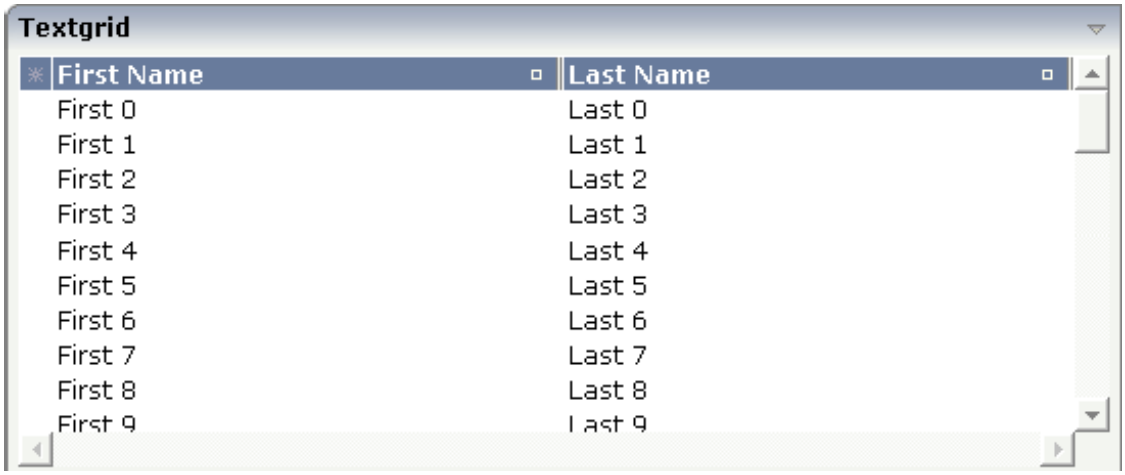
  There is a TEXTGRIDSSS2 control that is a certain variant of the TEXTGRID2 control.
- The ROWTABLEAREA2 is a container that internally allows you to use any normal control to be embedded inside a grid. Therefore, you can place normal FIELD controls, CHECKBOX controls etc. inside the ROWTABLEAREA2 container.

Use the TEXTGRID2 controls for displaying and selecting data. Use ROWTABLEAREA2 for entering data inside a grid.

59 TEXTGRID2

A Simple Example

The following example shows a TEXTGRID2 control:



There are two columns which hold data. There is one column at the very left which displays a selection icon - in addition to a yellow background for a selected line. Even and odd lines are displayed in slightly different colors. At the very right of each title column, there is a symbol which indicates the sorting status; if you double-click on this symbol, the column is sorted first in ascending direction and, when clicking again, in descending direction. Change the sequence of columns by dragging the title of a column and dropping it on another column's title. Depending from where you drop, the column is either moved left or right.

The asterisk in the upper left corner of the grid is used to select/deselect all lines in the grid. The behavior depends on the setting of the `singleselect` property which determines whether multiple lines can be selected in the grid (default) or whether only one line can be selected:

- **Multiple Line Selection Mode**
  When you choose the asterisk for the first time, all lines are selected. When you choose the asterisk a second time, all lines are deselected.
- **Single Line Selection Mode**
  When you choose the asterisk (no matter how often), an existing selected line is deselected.

The XML layout definition is:

```
<rowarea name="Textgrid">
    <itr takefullwidth="true" fixlayout="true">
        <textgrid2 griddataprop="lines" width="100%" height="200" ↩
selectprop="selected"
                   hscroll="true">
            <column name="First Name" property="firstName" width="50%">
            </column>
            <column name="Last Name" property="lastName" width="50%">
            </column>
        </textgrid2>
    </itr>
    <vdist height="5">
    </vdist>
</rowarea>
```

The TEXTGRID2 definition is bound to a grid data property `lines`. This is a special collection that mirrors the server data. Technically, it is treated in the same way as a normal collection. It supports the `Collection` and `List` interface.

Inside the TEXTGRID2 control definition there are two columns. These columns are bound to the properties `firstName` and `lastName`.

This is the Java adapter source:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.SelectableLine;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

// This class is a generated one.

public class TextGridAdapter
    extends Adapter
{
    // class >LinesItem<
    public class LinesItem
    extends SelectableLine
    {
        // property >firstName<
        String m_firstName;
        public String getFirstName() { return m_firstName; }

        // property >lastName<
        String m_lastName;
        public String getLastName() { return m_lastName; }

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

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

    /** initialisation - called when creating this instance*/
    public void init()
    {
        for (int i=0; i< 50; i++)
        {
            LinesItem l = new LinesItem();
            l.m_firstName = "First " + i;
            l.m_lastName = "Last " + i;
            m_lines.add(l);
        }
    }
}
```

The adapter class provides a property `lines`. This property returns an instance of the class `TEXTGRIDCollection` which itself is a special collection and comes with the Application Designer runtime. The instance is filled in the `init()` method of the adapter - just as a normal collection. But it automatically brings in all the functions for sorting and - if desired - server-side scrolling (see the **TEXTGRIDSSS2** description).

The collection is filled with objects of the inner class `Line`. Each object supports a property `firstName`, `lastName` and `selected`. (The class `Line` is an inner class in the example - but of course it could also be a normal class). Make sure to make the class publicly accessible, because the Application Designer runtime requires public access to the corresponding properties.

The whole TEXTGRID2 definition is bound by the `griddataprop` property to the `lines` collection - and each COLUMN definition is bound to a property of class `Line`, i.e. the class representing elements of the collection.

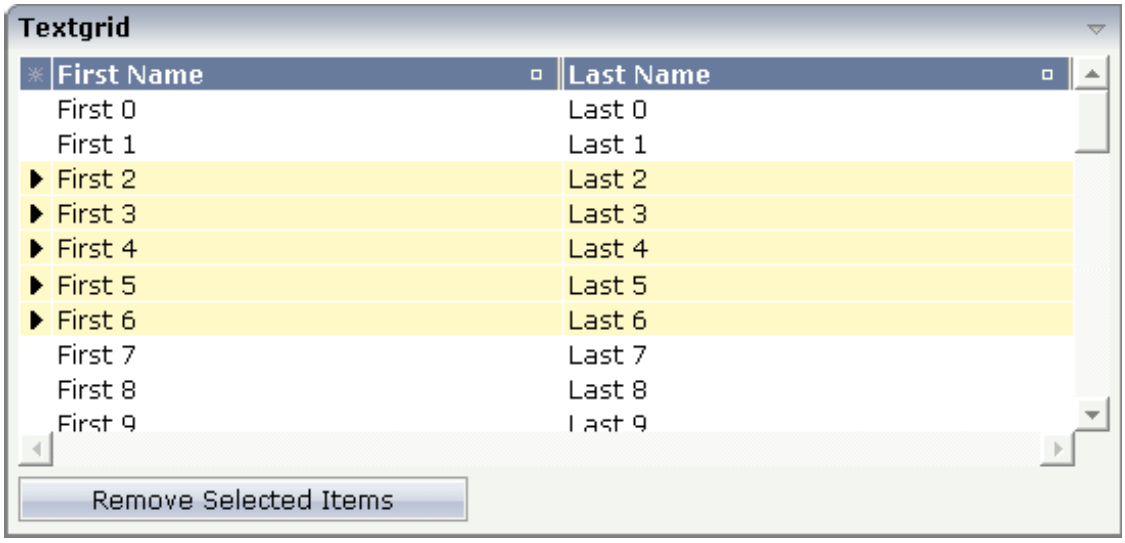
Selecting Rows in a TEXTGRID2

Maybe you wonder why there is a `selected` property in the class `Line` of the previous example.

This property is required for indicating which lines are currently selected and which are not. Each line which is displayed in the TEXTGRID2 control is represented at the server side by an object of the class `Line`. Therefore, the selection status of the grid (which lines are selected and which lines are not) is mirrored by the corresponding `selected` property of each line.

The code below shows an extension of the previous example. It demonstrates how to build a method for taking the line selection into consideration.

Below the TEXTGRID2 definition, there is a button that triggers a method for removing the selected lines.



The XML layout definition is improved in the following way:

```
<rowarea name="Textgrid">
    <itr takefullwidth="true" fixlayout="true">
        <textgrid2 griddataprop="lines" width="100%" height="200" ↩
selectprop="selected"
                   hscroll="true">
            <column name="First Name" property="firstName" width="50%">
            </column>
            <column name="Last Name" property="lastName" width="50%">
            </column>
        </textgrid2>
    </itr>
    <vdist height="5">
    </vdist>
    <itr>
        <button name="Remove Selected Items" method="onRemoveSelectedItems">
        </button>
    </itr>
    <vdist>
    </vdist>
</rowarea>
```

Note that inside the TEXTGRID2 definition, there is a property `selectprop` that points to the name of the item property used for storing the selection information accordingly.

The method `onRemoveSelectedItems` was added into the adapter code of the previous example:

```
public void onRemoveSelectedItems()
{
    for (int i=m_lines.size()-1; i>=0; i--)
    {
    LinesItem l = (LinesItem)m_lines.get(i);
    if (l.getSelected() == true)
    m_lines.remove(i);
}
```

The collection is iterated from its last element to its first. All elements which hold a `selected` property with value "true" are removed.

Note: In this example, you are able to select multiple rows inside the grid. If you want to allow selecting only one item, use the property `singleselect` inside the TEXTGRID2 definition.

Triggering Adapter Methods when Selecting a Row

In the previous section, you saw how to manage selections inside a TEXTGRID2 control. Sometimes, you want to trigger a certain function when selecting a row - maybe you want to react directly to the selected item.

To do so, you can use some additional properties inside the TEXTGRID2 definition:

- The `onclickmethod` property is used to point to a method of your adapter class which is called when a click event occurs.
- The `ondblclickmethod` property is used when the user double-clicks a grid row.

You can use “direct triggering of method” together with single line selection mode or with multiple line selection mode. In case of using it with multiple line selection, you have to find out which was the “last selected index”, i.e. the line index of the clicked/double-clicked line.

There is a property `lastselectedprop` inside the TEXTGRID2 definition. Using this definition, you can bind the value to an integer property of your adapter class. The index value which is selected is passed into this property.

TEXTGRID2 Properties

Basic			
griddataprop	Name of adapter property that represents the grid on server side. The property must be of type "TEXTGRIDCollection". var m_items = new TEXTGRIDCollection() Pay attention: once you have created an instance of TEXTGRIDCollection inside your adapter always exactly use this one instance. Do not re-instantiate collection objects! - Example: Instead of... WRONG: m_items = new TEXTGRIDCollection(); ...use... CORRECT: m_items.clear();	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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other	Obligatory	100 150 200 250

	controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Selection			
selectableprop	Name of the adapter parameter used for selectable property for the textgrid(textgridsss) control.	Optional	
selectprop	Name of property of the item objects - representing the individual rows of the text grid - that is used for selecting rows. Must be of type "boolean"/ "Boolean". If the user selects a text grid row then the value "true" is passed into the corresponding row object's property.	Optional	
singleselect	If set to "true" then only one row can be selected inside the text grid. - If set to "false" then multiple lines can be selected by using Ctrl- and Shift-key during mouse selection. Default is "false".	Optional	true false
singleselectprop	Name of adapter property that dynamically defined whether SINGLESELECT is true or false. Must return 'true' or 'false'.	Optional	
onclickmethod	Adapter method that is called when the user selects a row. Inside the adapter you can find the selected rows by iterating through the row objects and finding out which one's selection-property is switched to "true". In case of multiple row selection you can also use the method "findLastSelectedItem()" of your corresponding TEXTGRIDCollection object.	Optional	
ondblclickmethod	Adapter method that is called when the user selects a row by a double click.	Optional	

	Inside the adapter you can find the selected rows by iterating through the row objects and finding out which one's selection-property is switched to "true". In case of multiple row selection you can also use the method "findLastSelectedItem()" of your corresponding TEXTGRIDCollection object.		
withselectioncolumn	When defining a SELECTPROP property then automatically a selection column is added as first left column of the grid. Inside the column an icon indicates if a row is currently selected. Set this property to "false" in order to avoid the selection column.	Optional	true false
withselectioncolumnicon	Flag that indicates whether the selection column shows a "select all" icon on top. Default is true.	Optional	true false
fgselect	if switched to true then an additional "graying" of selected lines will be activated. Switch this property to "true" if you have coloured textgrid cells: the selection colour will not override the colour of each cell, as consequence you require an additional effect in order to make the user see which row is selected.	Optional	true false
focusedprop	Name of property of the item objects - representing the individual rows of the text grid - that indicates if the line should receive focus. Must be of type "boolean"/ "Boolean".	Optional	
focusable	Set this to TRUE if you want the browser to treat the grid like a focusable element (like buttons and input elements) when the page is loaded. Example usage: When your grid is the first focusable element the browser will automatically set the focus on the grid when the page is loaded.	Optional	true false
gridfocusedprop	Name of the property that indicates if the grid should receive the focus. It is automatically reset after the server-roundtrip.	Optional	
cursormgtprop	Only use this if you would like to reset the keyboard cursor from your Java program. Name of the java field that sets -1 in case the keyboard cursor should be reset after this server roundtrip.	Optional	
Right Mouse Button			
oncontextmenumethod	If clicking on a row of the text grid with the right mouse button then always the method "reactOnContexMenuRequest()" is called inside the corresponding row item object (that itself is kept inside the TEXTGRIDCollection object).	Optional	

	If the user clicks with the right mouse button onto an empty area of the grid then there is no object to call - instead the adapter method that is specified by this property is called.		
singleselectcontextmenu	With SHIFT and CTRL key the user can select multiple lines (use property SINGLESELECT to suppress this feature). Use this property to ensure that the context menu is requested only for a single line. Default is "false".	Optional	true false noselection
enabledefaultcontextmenu	Use this property to enable the default context menu of the browser within the textgrid. Please note: do not enable the browser's context menu if your application itself provides for a context menu. Default is "false".	Optional	true false
Appearance			
width	(already explained above)		
height	(already explained above)		
minapparentrows	Number of rows that are displayed independent of the size of the server side collection.	Optional	1 2 3 int-value
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Sometimes obligatory	auto scroll hidden
withtitlerow	If defined as "false" then no top title row is shown. "True" is default.	Optional	true false
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 synchronized within one container (i.e. TR, STR table	Optional	1 2 3 4 5 50

	rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
personalizable	If defined to "false" then no re-arranging of columns is offered to the user. Default is "true". This means: if using COLUMN controls inside the grid definition then the user can re-arrange the sequence of columns by dragging and dropping them within the top title row.	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2
stylevariantprop	Name of the adapter property which dynamically defines the STYLEVARIANT value at runtime.	Optional	VAR1 VAR2
backgroundstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080	Optional	

	You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "scroll".	Optional	auto scroll hidden
withrollover	The textgrid controls provide for a so called "roll over" effect. The row that is currently below the mouse pointer is highlighted in a certain way. Use this property to disable the roll over effect (Default is TRUE).	Optional	true false
fixedcolumnsizes	When switching the FIXEDCOLUMNSIZES property to value "true" then internally the grid is arranged in a way that the area always determines its size out of the width specification of the COLUMN controls. The browser does not look into the column contents in order to try to optimise the size of the area - but always follows the width that you define.	Optional	true false
requiredheight	Minimum height of the control in pixels. Use this property to ensure a minimum height if the overall control's height is a percentage of the available space - i.e. if value of property HEIGHT is a percentage (e.g. 100%). Please note:You must not use FIXLAYOUT at the surrounding row container (ITR and ROWAREA). Otherwise: if the available space is less than the required height the end of the control is just cut off.	Optional	1 2 3 int-value
disablecolumnresizing	Flag that indicates if the user can change the width of the grid columns. Default is false.	Optional	true false
disablecolumnmoving	Flag that indicates if the user can change the order of grid columns. 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
Drag And Drop			
draginfoprop	Name of the row item property that passes back the line's "drag info". When using this attribute the grid lines can be dragged onto "drop targets" (e.g. DROPICON control). The dragged line is identified by its "drag info". Use any string/information applicable.	Optional	
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	
Deprecated			
directselectevent	Use ONCLICKMETHOD and ONDBLCLICKMETHOD instead.	Optional	ondblclick onclick
directselectmethod	Use ONCLICKMETHOD and ONDBLCLICKMETHOD instead.	Optional	

COLUMN Properties

The COLUMN tag is the typical tag that is placed inside a TEXTGRID2 definition. The COLUMN definition defines a column with its binding to a property of the collection elements.

Tip: If you set the property `headernowrap="false"`, you usually have to increase the height of the header in the style sheet of your layout page. You can do this in the Style Sheet Editor: Go to the **Style Details** tab, expand the tree for TEXTGRID and then adjust the `height` value for `TEXTGRIDCellHeaderUnsorted`.

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	
property	Property of the row item object that represents the column's content. The content typically is straight text but can also be "complex HTML".	Obligatory	
width	Width of the control. There are two possibilities to define the width: (A) Pixel sizing: just input a number value (e.g. "100"). (B) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element (textgrid2, textgridsss2) of the control properly defines a width this control can reference.	Obligatory	100 120 140 160 180 200 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	date float int long time timestamp color xs:decimal

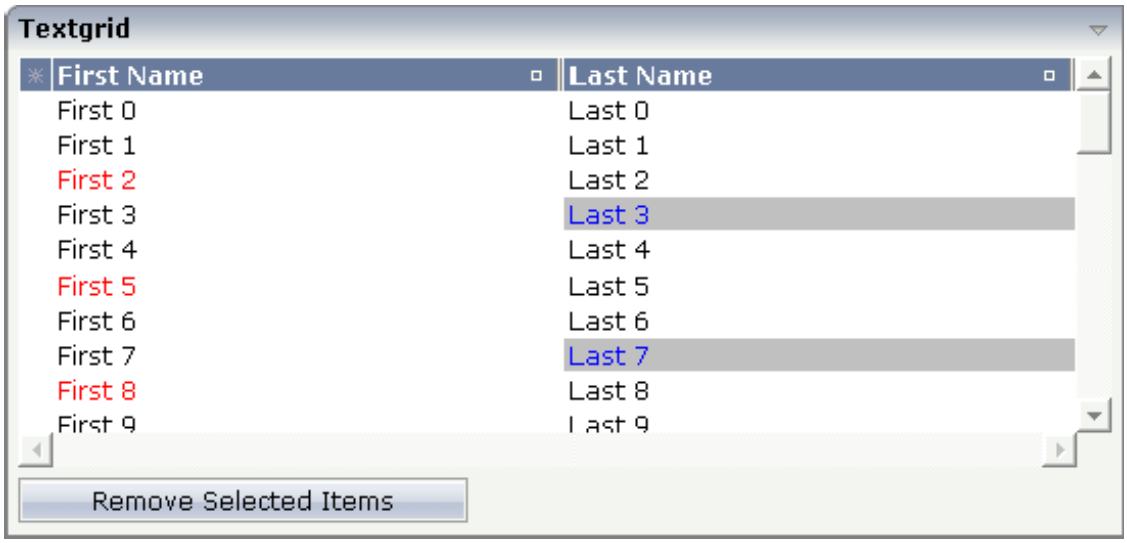
			xs:double xs:date xs:dateTime xs:time ---------- N n.n P n.n string n L xs:boolean xs:byte xs:short
align	Horizontal alignment of the control's content. Default is "center".	Optional	left center right
straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".	Optional	true false
convertspaces	If switched to "true" then all spaces inside the text that is rendered into the column are converted to non breakable spaces. Use this option if you have "meaningful" spaces inside the values you return from the server adapter object, e.g. if outputting some ASCII protocol inside a column.	Optional	true false
cuttextline	If switched to "false" then the content of the column is broken if it exceeds the column's width definition. Default is "true" i.e. if the content is too big for the column cell then it is cut.	Optional	true false
withsorticon	Flag that indicates if a small sort indicator is shown within the right corner of the control. Default is TRUE.	Optional	true false

headerimage	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. specifying "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	
headernowrap	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
Binding			
property	(already explained above)		
textstyleprop	Name of the property of the row item object that passes back a style-string that is used for rendering the column's content. As consequence you can indiviudally assign a CSS-style to each cell of your text grid.	Optional	
textclassprop	Name of the property of the row item object that defines a style class to be used for rendering the content. You can set up a limited number of style classes inside your style sheet definition - and dynamically reference them per grid cell.	Optional	
imageprop	Name of the property of the row item object passing back an image URL. The image is rendered at the very left of the column's area - in front of the text (PROPERTY property definition).	Optional	
linkmethod	Name of a method within the row item object that is called if user clicks the column's text.	Optional	
celllinkmethodprop	Name of the row item property that passes back the name of a method or null. If the method name is not null then the corresponding column (cells) will show the text as method link. On click the provided row item cell method is called.	Optional	
celltitleprop	Name of the property of the row item object that passes back the tooltip of this cell.	Optional	

Online help			
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 language management - representing the tooltip text that is used for the control.	Optional	
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 language management - representing the tooltip text for the sort indicator.	Optional	
celltitleprop	(already explained above)		

Dynamic Setting of Text Styles in TEXTGRID2

The example from the previous sections will now be enhanced in order to demonstrate how to control the style of cells inside a TEXTGRID2 control dynamically:



Some of the cells in the TEXTGRID2 control are rendered with a different style than the normal one. Each COLUMN definition has the property `textstyleprop`:

```
<rowarea name="Textgrid">
    <itr takefullwidth="true" fixlayout="true">
        <textgrid2 griddataprop="lines" width="100%" height="200" ↩
selectprop="selected"
                   hscroll="true">
            <column name="First Name" property="firstName" width="50%"
                    textstyleprop="firstNameStyle">
            </column>
            <column name="Last Name" property="lastname" width="50%"
                    textstyleprop="lastNameStyle">
            </column>
        </textgrid2>
    </itr>
    <vdist height="5">
    </vdist>
    <itr>
        <button name="Remove Selected Items" method="onRemoveSelectedItems">
        </button>
    </itr>
</rowarea>
```

The referenced property inside the COLUMN definition is on the same level as the normal property that is responsible for the content of the columns and which is referenced by the normal `property` property. Have a look at the Java source:

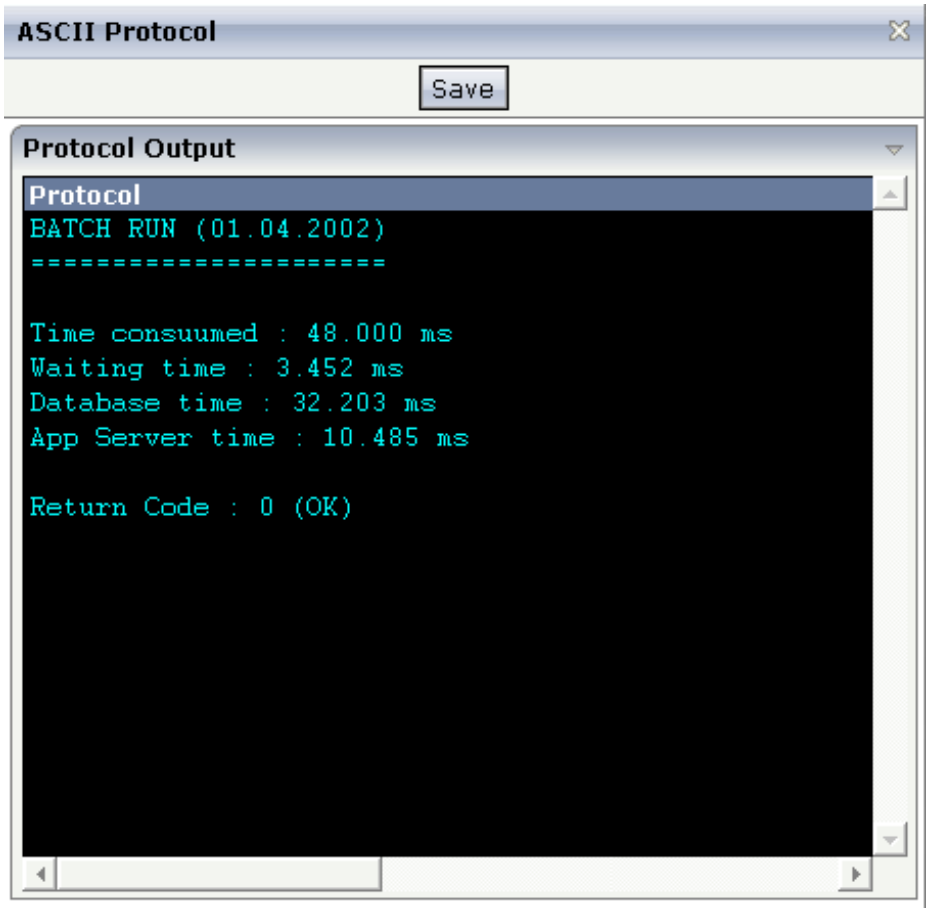
```
// This class is a generated one.

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.SelectableLine;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

// This class is a generated one.

public class TextGridAdapter
    extends Adapter
{
    // class >LinesItem<
    public class LinesItem
    extends SelectableLine
    {
        // property >firstName<
        String m_firstName;
        public String getFirstName() { return m_firstName; }

        // property >lastName<
        String m_lastName;
        public String getLastName() { return m_lastName; }

        // property >selected<
        boolean m_selected;
        public boolean getSelected() { return m_selected; }
```

```
        public void setSelected(boolean value) { m_selected = value; }

        // property >firstNameStyle<
        String m_firstNameStyle;
        public String getFirstNameStyle() { return m_firstNameStyle; }
        public void setFirstNameStyle(String value) { m_firstNameStyle = value; }

        // property >lastNameStyle<
        String m_lastNameStyle;
        public String getLastNameStyle() { return m_lastNameStyle; }
        public void setLastNameStyle(String value) { m_lastNameStyle = value; }
    }

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

    ...

    /** initialisation - called when creating this instance*/
    public void init()
    {
        for (int i=0; i< 50; i++)
        {
        LinesItem l = new LinesItem();
        l.m_firstName = "First " + i;
        l.m_lastName = "Last " + i;
        if (i%3 == 2)
            l.setFirstNameStyle("color: #FF0000;");
        if (i%4 == 3)
            l.setLastNameStyle("color: #0000FF; background-color: #C0C0C0");
        m_lines.add(l);
        }
    }
}
```

The properties `lastNameStyle` and `firstNameStyle` are available on item level. They are filled in the `init()` method.

Example: Displaying an ASCII Protocol

The following example shows the output of an ASCII protocol. The example demonstrates the usage of the COLUMN properties `textstyleprop` and `convertspaces`.



The XML layout definition looks as follows:

```
<page model="Ascii_Protocol_Adapter">
    <titlebar name="ASCII Protocol">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
        <rowarea name="Protocol Output">
            <itr takefullwidth="false" height="350" fixlayout="true">
                <textgrid2 griddataprop="items" width="100%" height="100%" ↩
hscroll="true"
                           vscroll="true" backgroundstyle="background-color:#000000">
                    <column name="Protocol" property="protocolText" width="1000"
                            textstyleprop="protocolStyle" convertspaces="true">
                    </column>
                </textgrid2>
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
```

```
    </statusbar>
</page>
```

The following is defined in the above layout definition:

- Inside the TEXTGRID2 definition, a black background is defined (`backgroundstyle` property).
- Inside the COLUMN definition, a style property is referenced (`textstyleprop` property).
- Inside the COLUMN definition, the property `convertspaces` is set to "true".

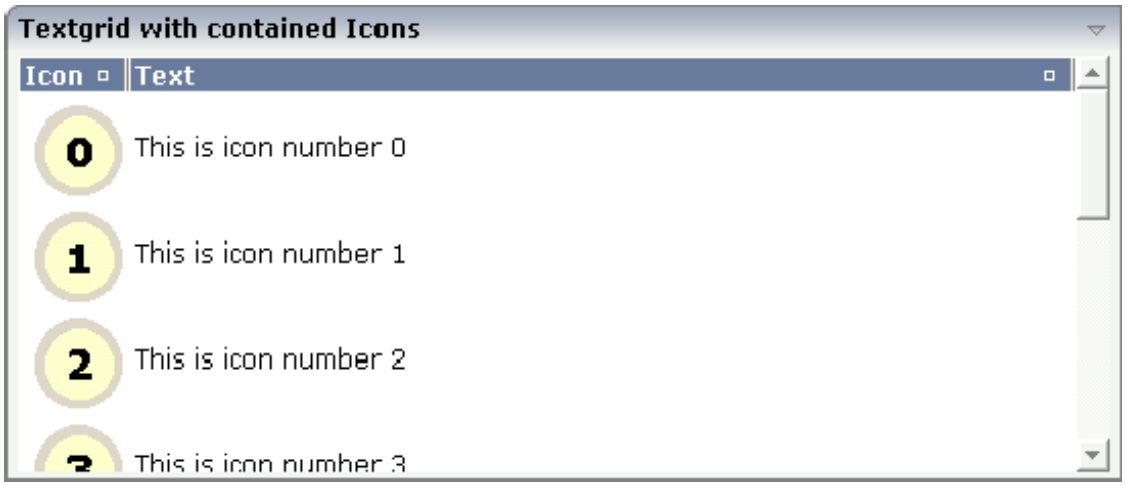
The Java source looks as follows:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

// This class is a generated one.

public class Ascii_Protocol_Adapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // inner classes
    // -----------------------------------------------------------------------

    // class >ItemsItem<
    public class Item
    {
        // property >protocolStyle<
        String m_centralStyle = "font-family: courier; color: #00FFFF; ↩
background-color: #000000;";
        String m_protocolStyle;
        public String getProtocolStyle() { return m_centralStyle; }

        // property >protocolText<
        String m_protocolText;
        public String getProtocolText() { return m_protocolText; }
        public void setProtocolText(String value) { m_protocolText = value; }
    }
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------

    // property >items<
    TEXTGRIDCollection m_items = new TEXTGRIDCollection();
    public TEXTGRIDCollection getItems() { return m_items; }

    // -----------------------------------------------------------------------
    // standard adapter methods
    // -----------------------------------------------------------------------

    /** initialisation - called when creating this instance*/
```

```
    public void init()
    {
        Item item;
        item = new Item(); item.setProtocolText("BATCH RUN (01.04.2002)"); ↩
m_items.add(item);
        item = new Item(); item.setProtocolText("======================"); ↩
m_items.add(item);
        item = new Item(); item.setProtocolText(""); m_items.add(item);
        item = new Item(); item.setProtocolText("Time consuumed : 48.000 ms");
        m_items.add(item);
        item = new Item(); item.setProtocolText("Waiting time : 3.452 ms");
        m_items.add(item);
        item = new Item(); item.setProtocolText("Database time : 32.203 ms");
        m_items.add(item);
        item = new Item(); item.setProtocolText("App Server time : 10.485 ms");
        m_items.add(item);
        item = new Item(); item.setProtocolText(""); m_items.add(item);
        item = new Item(); item.setProtocolText("Return Code : 0 (OK)"); ↩
m_items.add(item);
    }
}
```

Example: Using Images inside the TEXTGRID2 Control

In the following text grid, graphical information and text information are mixed:



The layout definition looks as follows:

```
<rowarea name="Textgrid with contained Icons">
    <itr takefullwidth="tue">
        <textgrid2 griddataprop="lines" width="100%" height="200">
            <column name="Icon" width="53" imageprop="iconURL">
            </column>
            <column name="Text" property="text" width="100%">
            </column>
        </textgrid2>
    </itr>
</rowarea>
```

In the definition of the left column, the property `imageprop` is used to reference to a property that provides the URL string of the image to be displayed. The definition of the right column contains the property `property` that points to a property providing text information.

The adapter class looks as follows:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

public class Textgrid_02_Adapter
    extends Adapter
    {
    // -----------------------------------------------------------------------
    // inner classes
    // -----------------------------------------------------------------------
    // class >LinesItem<
    public class LinesItem
    {
        // property >iconURL<
        String m_iconURL;
        public String getIconURL() { return m_iconURL; }
        public void setIconURL(String value) { m_iconURL = value; }

        // property >text<
        String m_text;
        public String getText() { return m_text; }
        public void setText(String value) { m_text = value; }
    }
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------
    // property >lines<
    TEXTGRIDCollection m_lines = new TEXTGRIDCollection();
    public TEXTGRIDCollection getLines() { return m_lines; }

    // -----------------------------------------------------------------------
    // standard adapter methods
    // -----------------------------------------------------------------------
```

```
/** initialisation - called when creating this instance*/
 public void init()
 {
     for (int i=0; i<10; i++)
     {
     LinesItem l = new LinesItem();
     l.setIconURL("images/touch_"+i+".gif");
     l.setText("This is icon number " + i);
     m_lines.add(l);
     }
 }
```

You can also mix text and image by specifying the `property` and the `imageprop` property. In this case, the image will be drawn on the left and the text will be placed to the right of the image.

Specifying the Width of a TEXTGRID2 Control

The sizing of text grids was improved with a previous release: now you can simply set a width of e.g. "100%" if the text grid should cover the complete width that is available.

Pay attention to the following:

- If you do not specify a width inside the TEXTGRID2 definition, the width of the grid will be as wide as defined by its content. Of course, it does not make sense to define a percentage value inside the COLUMN definitions - there is nothing to refer to.
- If you specify a width in the TEXTGRID2 and you already know that the size of the columns does not fit into the given width, you must set the flag `HSCROLL` to "true". Otherwise, there will be no scrolling at all and the grid will be rendered as wide as required by its content.
- If you specify a percentage value as a width for the TEXTGRID2 control, you must place the grid into an ITR definition that itself has also a WIDTH definition (typically of "100%"). In addition, you must set the flag `FIXLAYOUT` to "true" on ITR level. Otherwise the grid will follow the width of its contained columns.

Change Index Management

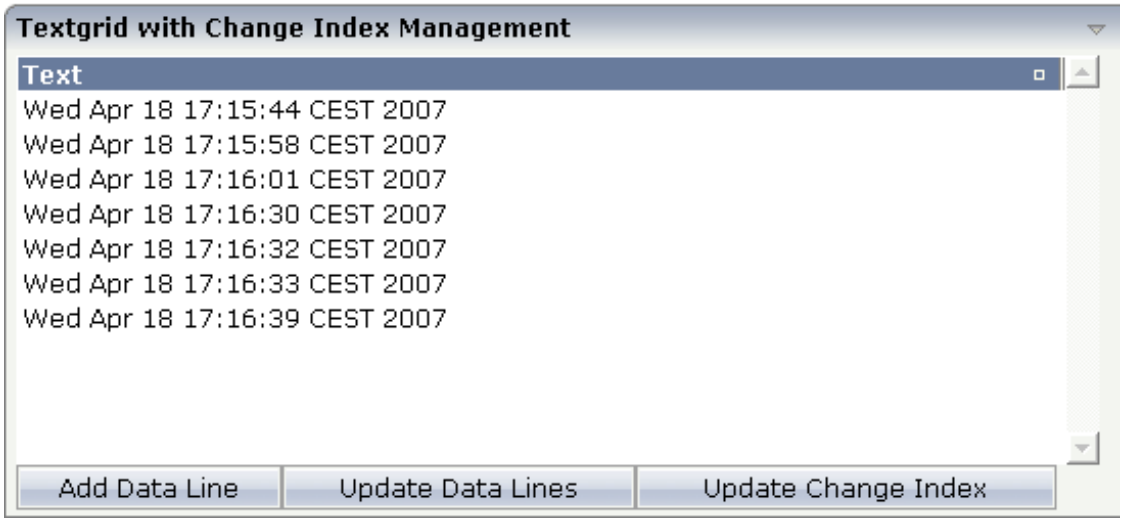
In order to improve performance on the client side, there is a so-called change index management: a text grid binds to an array of data records. Every time when the browser client receives updated data from the server, it finds out whether a text grid has to be updated or not. Updating a text grid is a quite expensive operation for the client - consequently, it is done only if really necessary.

For this reason, each `TEXTGRIDCollection` object implicitly administers a change index. A change index is a property of type `long`. The value of the property always changes if something inside

the collection changes. The client reads this property and only refreshes the text grid if the property has changed.

Normally, the property is managed internally - without you being involved. If a `TEXTGRIDCollection` is manipulated via its methods (e.g. `add` or `clear`), then the property is automatically updated - and consequently, the client refreshes. But: if a change of data happens inside one item of a `TEXTGRIDCollection`, then the call does not go through the `TEXTGRIDCollection` API. Consequently, you have to explicitly trigger the update by your program. Inside the `TEXTGRIDCollection`, there is a method `itemChanged()` which indicates that due to the change of data within one item the grid has to be updated.

The following example shows how to control the change index. In this example, a text grid is built and manipulated by three buttons:



With the first button, new items are added to the grid. With the second button, all items receive new content. With the third button, the change index will be updated.

The XML code is:

```
<rowarea name="Textgrid with Change Index Management">
    <itr>
        <textgrid2 griddataprop="lines_02" width="100%" height="200">
            <column name="Text" property="text" width="100%">
            </column>
        </textgrid2>
    </itr>
    <itr>
        <button name="Add Data Line" method="onAddDataLine">
        </button>
        <button name="Update Data Lines" method="onUpdateDataLines">
        </button>
        <button name="Update Change Index" method="onUpdateChangeIndex">
```

```
        </button>
    </itr>
</rowarea>
```

The Java adapter source is shown below. Pay attention to the constructor of the `m_lines` member (which passes "true" as a parameter) and to the method `m_lines.itemChanged()` that is called in order to update the change index implicitly.

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

import java.util.Date;
import java.util.Iterator;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

public class Textgrid_02_Adapter
    extends Adapter
{
    // -----------------------------------------------------------------------
    // inner classes
    // -----------------------------------------------------------------------
    // class >Lines_02Item<
    public class Lines_02Item
    {
        // property >text<
        String m_text;
        public String getText() { return m_text; }
        public void setText(String value) { m_text = value; }
    }
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------

    // property >lines_02<
    TEXTGRIDCollection m_lines_02 = new TEXTGRIDCollection();
    public TEXTGRIDCollection getLines_02() { return m_lines_02; }

    // -----------------------------------------------------------------------
    // public adapter methods
    // -----------------------------------------------------------------------
    /** */
    public void onAddDataLine()
    {
        Lines_02Item l = new Lines_02Item();
        l.setText((new Date()).toString());
        m_lines_02.add(l);
    }

    /** */
    public void onUpdateChangeIndex()
```

```
    {
        m_lines_02.itemChanged();
    }

    /** */
    public void onUpdateDataLines()
    {
        Iterator iter = m_lines_02.iterator();
        while (iter.hasNext())
        {
        Lines_02Item l = (Lines_02Item)iter.next();
        l.setText((new Date()).toString());
        }
    }
}
```

The behavior of the text grid control is as follows:

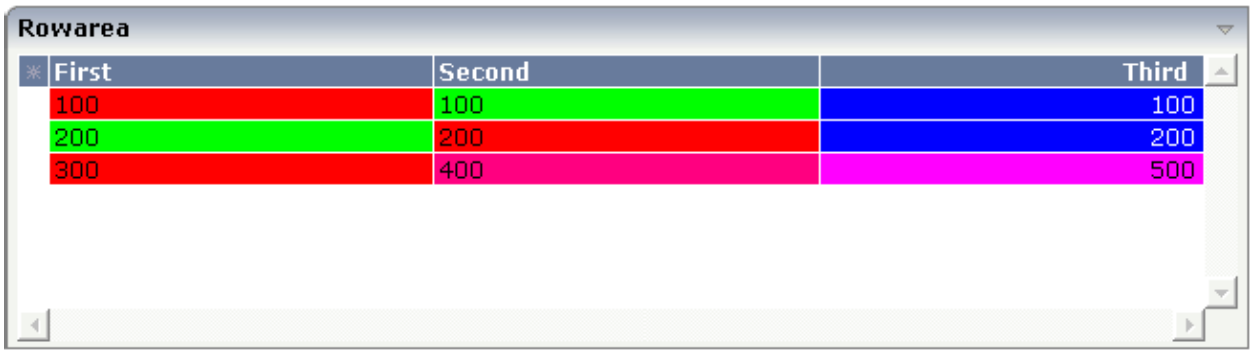
- If a new line is added (method `onAddDataLine()`), the change index will be updated internally - you do not have to explicitly tell the text grid management that something has changed.
- If the lines are updated (method `onUpdateDataLines()`), changes will not be reflected in the grid - until you explicitly trigger the method `onUpdateChanngeIndex()`.

Consequence: every time you change the inner content of the grid data, you have to update the change index by yourself.

Flexible Columns with CSVCOLUMN

There are situations in which the number and the format of the columns of a text grid cannot be defined in a fixed way inside the layout definition. The column type CSVCOLUMN allows you to dynamically define columns of a grid by your adapter program.

Have a look at the following example:



The control looks like a normal text grid. When looking inside the XML layout definition, you find out that instead of three fixed columns there is one dynamic column definition:

```
<rowarea name="Rowarea">
    <itr>
        <textgrid2 griddataprop="lines" width="100%" height="150" ↩
selectprop="selected"
                   singleselect="true" hscroll="true">
            <csvcolumn titlesprop="gridTitles" valuesprop="values" ↩
widthsprop="gridWidths"
                       alignsprop="gridAligns" backgroundsprop="backgrounds">
            </csvcolumn>
        </textgrid2>
    </itr>
</rowarea>
```

Inside the CSVCOLUMN definition, there is a binding to various properties that are provided for by the corresponding adapter:

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

import com.softwareag.cis.file.CSVManager;
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TEXTGRIDCollection;


public class textgrid_03_Adapter
    extends Adapter
{
    // ------------------------------------------------------------------------
    // inner classes
    // ------------------------------------------------------------------------
   // class >LinesItem<
    public class LinesItem
    {
        public LinesItem (String values, String backgrounds)
        {
            m_values = values;
            m_backgrounds = backgrounds;
        }

        // property >backgrounds<
        String m_backgrounds;
        public String getBackgrounds() { return m_backgrounds; }
        public void setBackgrounds(String value) { m_backgrounds = value; }

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

```
        // property >values<
        String m_values;
        public String getValues() { return m_values; }
        public void setValues(String value) { m_values = value; }
    }
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------
    String m_gridAligns = CSVManager.encodeString(new String[] {
            "left",
            "left",
            "right"
            });
    public String getGridAligns() { return m_gridAligns; }

    String m_gridTitles = CSVManager.encodeString(new String[] {
            "First",
            "Second",
            "Third"
            });
    public String getGridTitles() { return m_gridTitles; }

    String m_gridWidths = CSVManager.encodeString(new String[] {
            "200",
            "200",
            "200"
            });
     public String getGridWidths() { return m_gridWidths; }

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

    // -----------------------------------------------------------------------
    // public adapter methods
    // -----------------------------------------------------------------------
    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_lines.add(new LinesItem("100;100;100","#FF0000;#00FF00;#0000FF"));
        m_lines.add(new LinesItem("200;200;200","#00FF00;#FF0000;#0000FF"));
        m_lines.add(new LinesItem("300;400;500","#FF0000;#FF0080;#FF00FF"));
    }
}
```

The information for creating dynamic columns is passed as comma separated values. Comma separated values are either created directly as a string or by calling a static method of the class `com.softwareag.cis.file.CSVManager`.

Note: When using the `CSVManager` methods for creating comma separated value strings, this always pays attention to what happens if strings already include one or more semicolons.

Example: the `CSVManager` will encode the strings "A", "B1;B2" and "C" to "A;B1\;B2;C". On the client side, the "\;" is decoded back to ";".

Compare the layout definition with the code example in order to find out the exact binding technique between the control and the adapter properties.

CSVCOLUMN Properties

Tip: If you set the property `headernowrap="false"`, you usually have to increase the height of the header in the style sheet of your layout page. You can do this in the Style Sheet Editor: Go to the **Style Details** tab, expand the tree for TEXTGRID and then adjust the `height` value for `TEXTGRIDCellHeaderUnsorted`.

The properties of the CSVCOLUMN control are:

Basic			
titlesprop	Name of adapter property providing a semicolon-separated string containing the titles to be displayed. Example for a value that is passed back by the property: "First Name;Last Name;Street""	Obligatory	
valuesprop	Name of row item property that passes back the content of the cells - as semicolon-separated string.	Obligatory	
widthsprop	Name of adapter property provding a semicolon-separated string containing the widths of the columns to be displayed. Example for a value that is passed back by the property: "100;200;100%""	Obligatory	
alignsprop	Name of adapter property provding a semicolon-separated string containing the horizontal alignment of the columns to be displayed. Example for a value that is passed back by the property: "left\"center;right""	Sometimes obligatory	
backgroundsprop	Name of adapter property provding a semicolon-separated string containing the background color of the columns to be displayed. Example for a value that is passed back by the property: "\"#C0C0C0;#FF0000""	Optional	

proprefsprop	Name of adapter property provding a semicolon-separated string containing the row item properties that are internally used to build up the value string. The property names are used for sorting: if the user invoke the sorting of the grid by clicking on the corresponding icons inside the title cell then this column needs to be associated with an internal property that is used for sorting. Example: "firstName\"lastName;street""	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
straighttext	If the text of the control contains HTML tags then these are by default interpreted by the browser. specifying STRAIGHTTEXT as "true" means that the browser will directly render the characters without HTML interpretation. Example: if you want to output the source of an HTML text then STRAIGHTTEXT should be set to "true".	Optional	true false
cuttextline	If switched to "false" then the content of the column is broken if it exceeds the column's width definition. Default is "true" i.e. if the content is too big for the column cell then it is cut.	Optional	true false
headernowrap	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
withgridcolheaders	Flag that indicates if the user can resize column widths and re-order columns by drag and drop. Default is false. If set to true the corresponding adapter program must register as "column change event" listener. Use method TEXTGRIDCollection.registerGridColHeaderChangeListener for that.	Optional	true false
Binding			
textstyleprop	Name of the property of the row item object that passes back a style-string that is used for rendering the column's content. As consequence you can indiviudally assign a CSS-style to each cell of your text grid.	Optional	
textclassprop	Name of the property of the row item object that defines a style class to be used for rendering the content. You can set up a limited number of style classes inside your style sheet definition - and dynamically reference them per grid cell.	Optional	
straighttextprop	Name of the property which dynamically defines whether STRAIGHTTEXT is true or false.	Optional	

sorttitlesprop	Name of adapter property provding a semicolon-separated string containing the titles to be displayed. Example for a value that is passed back by the property: "Click here to sort column First Name\" Click here to sort column Last Name; Click here to sort column Street""	Optional	
tooltiptitlesprop	Name of adapter property provding a semicolon-separated string containing the tooltip tip texts to be displayed when the mouse is moved over the column headers.	Optional	
linkmethodsprop	Name of the property of the row item object that passes back (comma separated) names of row item methods. The corresponding columns will show the text as method links. On click the provided row item method is called.	Optional	
celllinkmethodsprop	Name of the row item property that passes back (comma separated) names of cell methods. The corresponding columns (cells) will show the text as method links. On click the provided row item cell method is called.	Optional	
celltooltiptitleprop	Name of the property of the row item object that passes back (comma separated) tool tip titles. The titles will show up if the user is moving slowly the mouse over the grid cells.	Optional	
imageprop	Name of the property of the row item object that passes back (comma separated) image URLs. The URL must either be an absolute URL or a relative URL.	Optional	
headerimageprop	Name of the property that passed back (comma separated) image URLs. The images are applied to the header.	Optional	

60 TEXTGRIDSSS2 - TEXTGRID2 with Server-Side Scrolling

The TEXTGRIDSSS2 control is a variant of the **TEXTGRID2** control which is explained in the previous section. "SSS" is the abbreviation for "server-side scrolling".

Performance Considerations

The TEXTGRID2 control fetches all items belonging to the grid and renders them according to its layout definition. If there are more items available than the grid can display, a vertical scroll bar is displayed and you can scroll through the list.

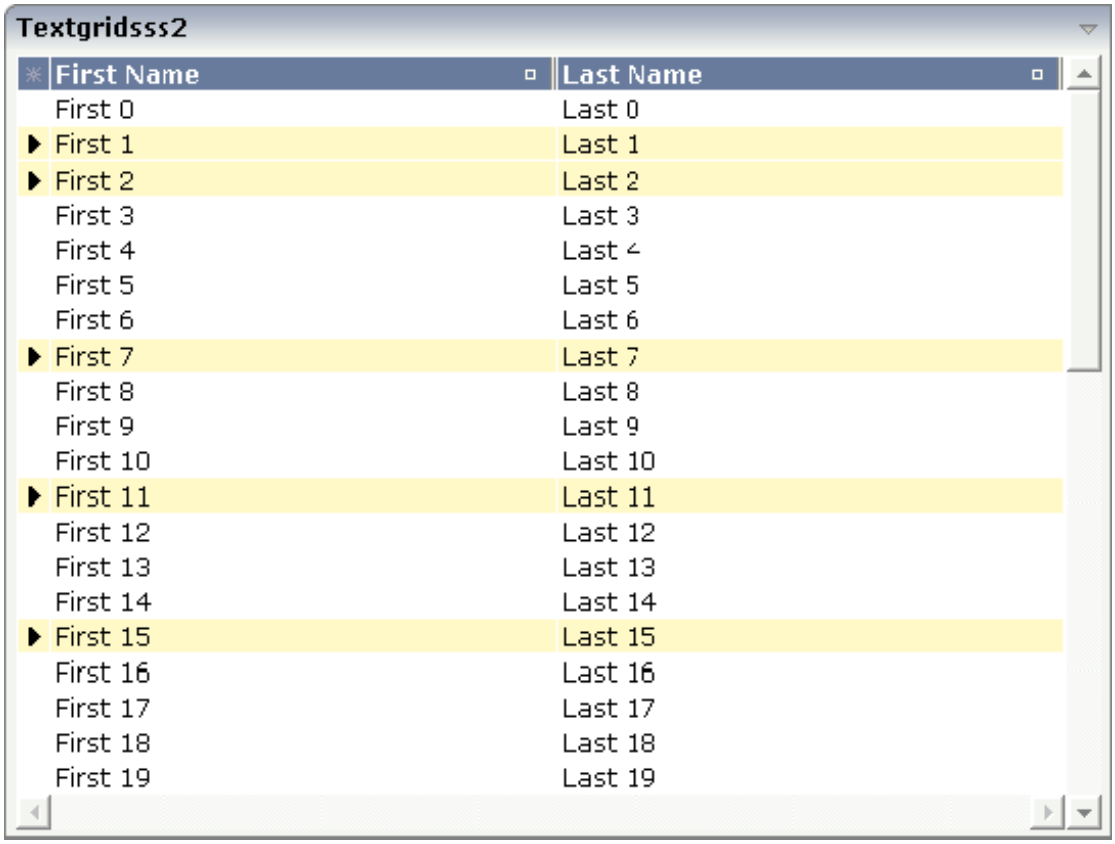
From scrolling perspective, this is very effective - the browser is very fast when scrolling is needed. But there are two disadvantages, especially for long lists:

- All the data that are to be displayed inside the grid must be available on the client side. Therefore, the data must be transferred from the server to the client at least one time. Imagine you have a grid of 10,000 lines: even if Application Designer transfers only “net data” and even if this happens in “delta transfer mode”, it must be transferred.
- In addition, the grid must be built completely in order to allow fast scrolling. This means - taking the above example - that 10,000 lines have to be rendered before the grid can be displayed. Table rendering is time-consuming and needs a lot of the client's CPU performance.

Consequence: text grids of the TEXTGRID2 control are easy to use, but they have their limitations in terms of scalability. You should use it only if a limited amount of information is to be displayed.

Example

The TEXTGRIDSSS2 is very similar to the TEXTGRID2 control it is even based on the same code. However, some special behavior has been built in. The main differences are “in the background”. The TEXTGRIDSSS2 control only receives the data of the visible items. In this example, only the data of the first 20 items are returned and rendered. When scrolling down, the next 20 items are fetched and rendered. This means: the control requests always the data which are currently displayed.



Consequence: every scrolling step requires an interaction with the server. However, only a small amount of data - which is visible - is requested, not the data of all available items. The performance of the grid does not change with the number of items which are available. There is no time difference in rendering a text grid containing 100 or 10,000 items.

The layout definition is:

```
<rowarea name="textgridsss2">
    <itr>
        <textgridsss2 griddataprop="lines" rowcount="20" width="100%"
                      selectprop="selected" singleselect="false" hscroll="true"
                      directselectmethod="onDirectSelection"
                      directselectevent="ondblClick">
            <column name="First Name" property="firstname" width="50%">
            </column>
            <column name="Last Name" property="lastname" width="50%">
            </column>
        </textgridsss2>
    </itr>
</rowarea>
```

The definition is nearly the same as for the TEXTGRID2 control - with the exception that there is a property `rowcount` to be used. The property `rowcount` defines the number of rows that are fetched from the server. All the other properties are the same as with TEXTGRID2.

No Change in Adapter Code between TEXTGRID2 and TEXTGRIDSSS2

No changes are required for the adapter source. You can use the same adapter code for TEXTGRID2 as for TEXTGRIDSSS2 controls. The server-side scrolling is completely done by the `TEXTGRIDCollection` class, already explained in section **TEXTGRID2**.

Note: In older versions you had to program server-side scrolling by yourself, and the code was not the same between TEXTGRID and TEXTGRIDSSS.

Using rowcount and height

Maybe you have noticed that in the TEXTGRIDSSS2 control, there are two properties for defining the height of the text grid: `rowcount` and `height`. The usage of the properties is as follows:

- `rowcount="20", height=""` (undefined)

  The text grid is rendered with exactly 20 lines. The height of the text grid is the height of the 20 lines.

- `rowcount="20", height="100%"` (or pixel value)

  The height is determined by the `height` definition. The text grid will only show these items which fit into the height. If the height only allows 12 lines to be shown, server-side scrolling always picks 12 items from the server instead of 20 as defined. The `rowcount` property defines the maximum number of items to be picked, i.e. if the height allows 25 lines to be displayed, only the maximum of 20 are picked.

By using `rowcount` and `height` together, you can have both server-side scrolling and dynamic vertical sizing of a text grid.

Setting the Client-Side Loading Behavior

As an alternative to server-side scrolling, you can customize the client-side loading behavior. Setting the property `onloadbehaviour="collection"` activates performance-optimized client-side scrolling in the client. As with the TEXTGRID2 control, the application must pass all items to the client at the beginning. But other than with the TEXTGRID2 control, these items are not immediately rendered in the grid. Instead, the client caches the items and only renders the items that are currently visible. If you have a limited number of items, the TEXTGRIDSSS2 control thus combines the advantage of the easy-to-use TEXTGRID2 control with better performance. For a really large number of items, however, server-side scrolling is still the best solution.

Sometimes, the number of items is not yet known at design-time. In such a case, you can set `onloadbehaviour="collectionorblock"`. Up to a defined number of items, the behavior is identical to `onloadbehaviour="collection"`. The maximum number of items for which client-side scrolling is to be done can be specified in the *cisconfig.xml* file. For a higher number of items, server-side scrolling is done. Switching between client-side scrolling and server-side scrolling is done automatically. You need not program anything to make it work.

TEXTGRIDSSS2 Properties

Basic			
griddataprop	Name of adapter property that represents the grid on server side. The property must be of type "TEXTGRIDCollection". var m_items = new TEXTGRIDCollection() Pay attention: once you have created an instance of TEXTGRIDCollection inside your adapter always exactly use this one instance. Do not re-instantiate collection objects! - Example: Instead of... WRONG: m_items = new TEXTGRIDCollection(); ...use... CORRECT: m_items.clear();	Obligatory	
rowcount	Number of rows that are rendered inside the control. There are two ways of using this property - dependent on whether you in addition define the HEIGHT property: If you do NOT define the HEIGHT property then the control is rendered with exactly the number of rows that are defined as ROWCOUNT value. If a HEIGHT value is defined in addition (e.g. as percentage value "100%") then the number of rows depend on the actual height of the control. The ROWCOUNT value in this case indicates the maximum number of rows that are picked from the server. You should define this value in a way so that it is not too low - otherwise your grid will not be fully filled. On the other hand it	Obligatory	

	should not be defined too high ("100") because this causes more communication traffic and more rendering effort inside the browser.		
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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%

onloadbehaviour	DEPRECATED: Loading behaviour of the items into the client. "block" (=default). Only "block" is supported´. All other settings are automatically mapped to "block".	Optional	block collection collectionorblock
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Selection			
selectableprop	Name of the adapter parameter used for selectable property for the textgrid(textgridsss) control.	Optional	
selectprop	Name of property of the item objects - representing the individual rows of the text grid - that is used for selecting rows. Must be of type "boolean"/ "Boolean". If the user selects a text grid row then the value "true" is passed into the corresponding row object's property.	Optional	
singleselect	If set to "true" then only one row can be selected inside the text grid. - If set to "false" then multiple lines can be selected by using Ctrl- and Shift-key during mouse selection. Default is "false".	Optional	true false
singleselectprop	Name of adapter property that dynamically defined whether SINGLESELECT is true or false. Must return 'true' or 'false'.	Optional	
onclickmethod	Adapter method that is called when the user selects a row. Inside the adapter you can find the selected rows by iterating through the row objects and finding out which one's selection-property is switched to "true". In case of multiple row selection you can also use the method "findLastSelectedItem()" of your corresponding TEXTGRIDCollection object.	Optional	
ondblclickmethod	Adapter method that is called when the user selects a row by a double click. Inside the adapter you can find the selected rows by iterating through the row objects and finding out which one's selection-property is switched to "true". In case of multiple row selection you	Optional	

	can also use the method "findLastSelectedItem()" of your corresponding TEXTGRIDCollection object.		
dblclickonreturn	If set to TRUE, the RETURN key will trigger the same method as an double click.	Optional	
withselectioncolumn	When defining a SELECTPROP property then automatically a selection column is added as first left column of the grid. Inside the column an icon indicates if a row is currently selected. Set this property to "false" in order to avoid the selection column.	Optional	true false
withselectioncolumnicon	Flag that indicates whether the selection column shows a "select all" icon on top. Default is true.	Optional	true false
fgselect	if switched to true then an additional "graying" of selected lines will be activated. Switch this property to "true" if you have coloured textgrid cells: the selection colour will not override the colour of each cell, as consequence you require an additional effect in order to make the user see which row is selected.	Optional	true false
focusedprop	Name of property of the item objects - representing the individual rows of the text grid - that indicates if the line should receive focus. Must be of type "boolean"/ "Boolean".	Optional	
focusable	Set this to TRUE if you want the browser to treat the grid like a focusable element (like buttons and input elements) when the page is loaded. Example usage: When your grid is the first focusable element the browser will automatically set the focus on the grid when the page is loaded.	Optional	true false
gridfocusedprop	Name of the property that indicates if the grid should receive the focus. It is automatically reset after the server-roundtrip.	Optional	
cursormgtprop	Only use this if you would like to reset the keyboard cursor from your Java program. Name of the java field that sets -1 in case the keyboard cursor should be reset after this server roundtrip.	Optional	
Right Mouse Button			
oncontextmenumethod	If clicking on a row of the text grid with the right mouse button then always the method "reactOnContexMenuRequest()" is called inside the corresponding row item object (that itself is kept inside the TEXTGRIDCollection object).	Optional	

	If the user clicks with the right mouse button onto an empty area of the grid then there is no object to call - instead the adapter method that is specified by this property is called.		
singleselectcontextmenu	With SHIFT and CTRL key the user can select multiple lines (use property SINGLESELECT to suppress this feature). Use this property to ensure that the context menu is requested only for a single line. Default is "false".	Optional	true false noselection
enabledefaultcontextmenu	Use this property to enable the default context menu of the browser within the textgrid. Please note: do not enable the browser's context menu if your application itself provides for a context menu. Default is "false".	Optional	true false
Appearance			
width	(already explained above)		
height	(already explained above)		
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "hidden".	Optional	auto scroll hidden
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "scroll".	Optional	auto scroll hidden
touchpadinput	Boolean property that decides if touch pad support is offered for the TEXTGRID control. The default is "false". If switched to "true" then you can scroll the grid via a touch pad. As consequence you can use this control for making inputs through a touch terminal.	Optional	true false

withtitlerow	If defined as "false" then no top title row is shown. "True" is default.	Optional	true false
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
personalizable	If defined to "false" then no re-arranging of columns is offered to the user. Default is "true". This means: if using COLUMN controls inside the grid definition then the user can re-arrange the sequence of columns by dragging and dropping them within the top title row.	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and	Optional	

	"VAR2" but does not predefine any semantics behind - this is up to you!		
stylevariantprop	Name of the adapter property which dynamically defines the STYLEVARIANT value at runtime.	Optional	VAR1 VAR2
backgroundstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
withblockscrolling	If switched to "true" then the grid will show small scroll icons by which the user can scroll the grid's content. Scrolling typically is done by using the grid's scrollbar - the scroll icons that are switched on by this property are an additional possibility to scroll.	Optional	true false
withrollover	The textgrid controls provide for a so called "roll over" effect. The row that is currently below the mouse pointer is highlighted in a certain way. Use this property to disable the roll over effect (Default is TRUE).	Optional	true false
fixedcolumnsizes	When switching the FIXEDCOLUMNSIZES property to value "true" then internally the grid is arranged in a way that the area always determines its size out of the width specification of the COLUMN controls. The browser does not look into the column contents in order to try to optimise the size of the area - but always follows the width that you define.	Optional	true false
requiredheight	Minimum height of the control in pixels. Use this property to ensure a minimum height if the overall control's height is a percentage of the available space - i.e. if value of property HEIGHT is a percentage (e.g. 100%).	Optional	1 2 3

	Please note:You must not use FIXLAYOUT at the surrounding row container (ITR and ROWAREA). Otherwise: if the available space is less than the required height the end of the control is just cut off.		int-value
minapparentrows	Minimum number of apparent rows. Insert a valid number to make sure that (e.g. 10) rows are shown for sure.	Optional	1 2 3 int-value
disablecolumnresizing	Flag that indicates if the user can change the width of the grid columns. Default is false.	Optional	true false
disablecolumnmoving	Flag that indicates if the user can change the order of grid columns. 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
removeemptylines	Only supported in NATPAGE layouts. When setting to TRUE, empty lines will be removed from the item array passed to NJX. The reduced array will be rendered in the browser. An empty line is a line which contains only empty alphanumeric fields or numeric fields which are 0.	Optional	true false
withsliderfreeze	Setting this to "true" prevents unwanted slider jumps while scrolling up/down in a grid with a huge number of lines (for example 20000).	Optional	true false
Drag And Drop			
draginfoprop	Name of the row item property that passes back the line's "drag info". When using this attribute the grid lines can be dragged onto "drop targets"	Optional	

	(e.g. DROPICON control). The dragged line is identified by its "drag info". Use any string/information applicable.		
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	
Deprecated			
directselectmethod	Use ONCLICKMETHOD and ONDBLCLICKMETHOD instead.	Optional	
directselectevent	Use ONCLICKMETHOD and ONDBLCLICKMETHOD instead.	Optional	ondblclick onclick
showemptylines	If set to false, no empty line will be rendered. By default empty lines are shown.	Optional	true false

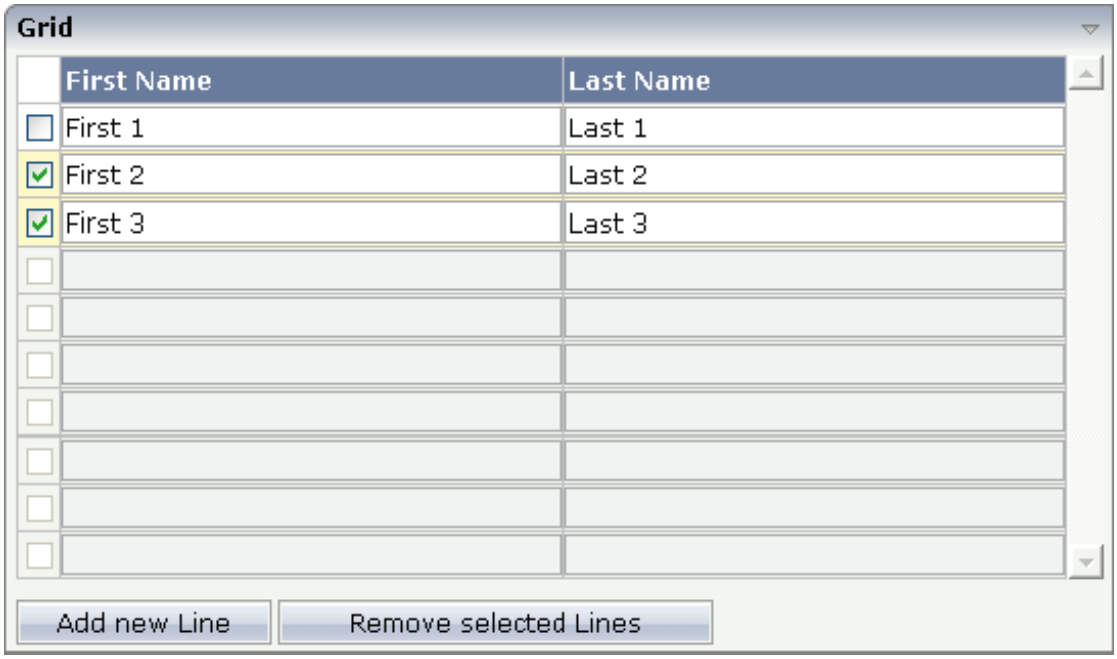
Inside the TEXTGRIDSSS2 definitions, COLUMN tags are also used to define its content. There is no difference in COLUMN tag usage between TEXTGRIDSSS2 and TEXTGRID2 definition.

61 ROWTABLEAREA2 - The Flexible Control Grid

The ROWTABLEAREA2 is a container control that allows other controls to be arranged inside its grid management.

Example for ROWTABLEAREA2

There is a grid that contains a header row and 10 lines. Each line contains one check box and two fields. Some of the lines are highlighted.



The XML layout definition is:

```
<rowarea name="Grid">
   <rowtablearea2 griddataprop="lines" rowcount="10" width="100%" withborder="true">
        <tr>
            <hdist>
            </hdist>
            <label name="First Name" asheadline="true">
            </label>
            <label name="Last Name" asheadline="true">
            </label>
        </tr>
        <repeat>
            <str valueprop="selected">
                <checkbox valueprop="selected" flush="screen" width="30">
                </checkbox>
                <field valueprop="firstname" width="50%">
                </field>
                <field valueprop="lastname" width="50%">
```

```
                </field>
            </str>
        </repeat>
    </rowtablearea2>
    <vdist height="10">
    </vdist>
    <itr>
        <button name="Add new Line" method="onAddLine">
        </button>
        <hdist>
        </hdist>
        <button name="Remove selected Lines" method="onRemoveLines">
        </button>
    </itr>
</rowarea>
```

Note the following:

- There is a ROWTABLEAREA2 definition with the property `griddataprop="lines"`. There is a `rowcount` definition of "10". This is the same as for the text grid processing: the grid container is bound to a server-side collection. Similar to the TEXTGRIDSSS2 definition, there is a row count that defines the number of lines.
- Inside the ROWTABLEAREA2 definition, there is first the definition of a normal table row (TR) in which a distance and two labels are defined. The labels are rendered with `asheadline="true"`.
- Inside the REPEAT definition, there is a special table row definition "STR" (selectable table row) that itself contains one CHECKBOX and two FIELD definitions. CHECKBOX and FIELDs are bound to properties themselves.
- After the ROWTABLEAREA2 definition, there is a vertical distance and a row that contains two buttons with which a user can manipulate the grid.

The content of the REPEAT block is repeated as many times as defined inside the `rowcount` definition of ROWTABLEAREA2. The content holds a table row (STR) - therefore the result is a grid.

The Java code of the adapter is:

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

import java.util.Iterator;
import java.util.Vector;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.GRIDCollection;

public class rowtabarea2_adapterAdapter
    extends Adapter
{
    // ---------------------------------------------------------------------
```

```
    // inner classes
    // ------------------------------------------------------------------------
    // class >LinesItem<
    public class LinesItem
    {
        // property >firstname<
        String m_firstname;
        public String getFirstname() { return m_firstname; }
        public void setFirstname(String value) { m_firstname = value; }

        // property >lastname<
        String m_lastname;
        public String getLastname() { return m_lastname; }
        public void setLastname(String value) { m_lastname = value; }

        // property >selected<
        boolean m_selected;
        public boolean getSelected() { return m_selected; }
        public void setSelected(boolean value) { m_selected = value; }
    }
    // ------------------------------------------------------------------------
    // property access
    // ------------------------------------------------------------------------
    // property >lines<
    GRIDCollection m_lines = new GRIDCollection();
    GRIDCollection getLines() { return m_lines; }

    // ------------------------------------------------------------------------
    // public adapter methods
    // ------------------------------------------------------------------------
    /** */
    public void onAddLine()
    {
        LinesItem l = new LinesItem();
        m_lines.add(l);
    }

    /** */
    public void onRemoveLines()
    {
        Vector v = new Vector();
//      collect all elements to be deleted
        Iterator iter = m_lines.iterator();
        while (iter.hasNext())
        {
            LinesItem l = (LinesItem)iter.next();
            if (l.getSelected() == true)
                v.addElement(l);
        }
//      delete elements
        iter = v.iterator();
        while (iter.hasNext())
```

```
            m_lines.remove(iter.next());
    }
}
```

Programming the grid is very simple. Define an instance of the class `GRIDCollection` in which you hold the items. This instance is referenced by the `griddataprop` definition inside the ROWTABLEAREA2 tag.

Each element inside the collection itself supports the properties that are referenced by the controls inside the REPEAT block. In our example, the properties are referenced by the STR, CHECKBOX and FIELD controls.

Use any “normal controls” inside the REPEAT block. For example, use either the BUTTON control or the ICON control. Properties of these included controls are called inside the item class `lines` and not directly in the adapter class.

The class `GRIDCollection` is the parent of `TEXTGRIDCollection`. It manages all aspects of server-side scrolling that is used internally.

Using rowcount and height

Similar to the TEXTGRIDSSS2 control, the ROWTABLEAREA2 controls offers two properties for defining its height:

- `rowcount` and
- `height`

If only `rowcount` is defined, the control will be always rendered with exactly the same number of lines - the one defined by the `rowcount` property.

If the `height` is specified additionally, the height of the grid will follow the `height` definition. The number of rows consequently follows the available vertical space. In this case, `rowcount` is the maximum number of rows that is exchanged.

Background information: if you have a look at the generated HTML page for an XML layout definition containing a ROWTABLEAREA2 grid, then you will see that each row of the grid is rendered into corresponding controls. If a ROWTABLEAREA2 contains 10 lines where each line has three FIELD controls, then the result will be an HTML page containing 30 fields.

Consequently, the `rowcount` property - when also specifying the `height` - should be carefully selected. You must not simply define a `rowcount` of "100" because then you will get very large HTML pages that become too large to be operated in a fast way. On the other hand, the `rowcount` should fit into normal screen sizes and should not be too low. Have in mind the screen sizes of your users and decide accordingly.

When does it make sense to have a `height` and a `rowcount` definition? Typically, it does not make sense to define a fixed height (for example, "200") for ROWTABLEAREA2 controls: instead of defining fixed heights, you should size the grid by using `rowcount` only. But it makes sense if you have flexible heights, for example, a height of "100%". In this case, the actual height depends on the size of the user's screen and the grid can thus be sized in a flexible way.

Making Grids Look like Grids

Fields typically contain a high number of FIELD controls. Typically, a FIELD control has a certain rendering that renders a field with a border and with a certain background color.

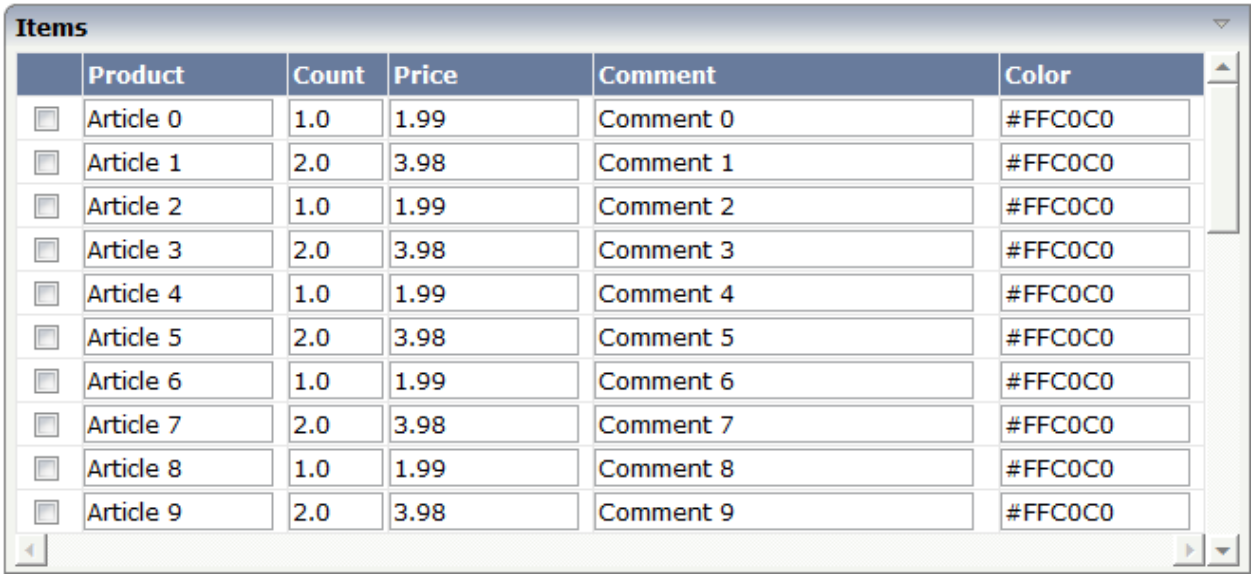
Be aware that inside the FIELD definition, there are two important properties:

- `noborder` - if set to "true", no border will be drawn
- `transparentbackground` - if set to "true", the field will always take over the background of the controls in which it is positioned (e.g. STR row).

Have a look at the difference between the following screens. One screen uses the properties, the other screen does not use them.

This is a grid:

This is collection of fields:



For information on how to build the lines of a grid dynamically, see the description of the **FLEXLINE** control.

Exported Columns

The dynamic values of the controls in the grid columns are exported. Columns with the following controls are exported:

- field
- checkbox
- combodyn2
- dateinput
- text
- textout
- label
- methodlink
- radiobutton
- combofix

Additional static text, for example, a label with a static name property, is not exported. Columns which have been set to invisible via the GRIDCOLHEADER control are not exported.

Export of Grid Headers

Whether the grid headers are exported or not can be set dynamically at runtime. It can be set from within the Natural program, or the end-users can toggle the export dynamically via a corresponding icon. Headers which have been set to invisible via the GRIDCOLHEADER control are not exported.

Exported Grid Items

The following functions are supported.

- Copy all Grid Items to the Clipboard
- Copy the Selected Items to the Clipboard
- Export all Grid Items to a File
- Export the selected items to a file

Copy all Grid Items to the Clipboard

All items are copied to the clipboard. In some versions of Firefox, an additional confirmation pop-up opens, asking you to allow access to the clipboard. In Chrome and Internet Explorer 11/Edge this additional pop-up is not shown.

Copy the Selected Items to the Clipboard

The selected items are copied to clipboard. Users can select items via the normal multiselect functionality of ROWTABLEAREA2.

Export all Grid Items to a File

All items are written to a *.csv* (comma separated values) file. The browser's standard dialog for opening or saving a file opens.

Export the selected items to a file

The selected items are written to a *.csv* file. Users can select items via the normal multiselect functionality of ROWTABLEAREA2.

Enabling Grid Export

Either use the properties `withcopytoclipboard`, `withselectedtoclipboard`, `withexporttofile`, `withselectedtofile` and `withexportheaders` to add icons for these functions to your grid, or use the ROWTABLEAREA2WITHBARS control as explained in the chapter "Icon Bars" below. The ROWTABLEAREA2WITHBARS supports a more flexible way to arrange the icons.

Special Events in ROWTABLEAREA2 Processing

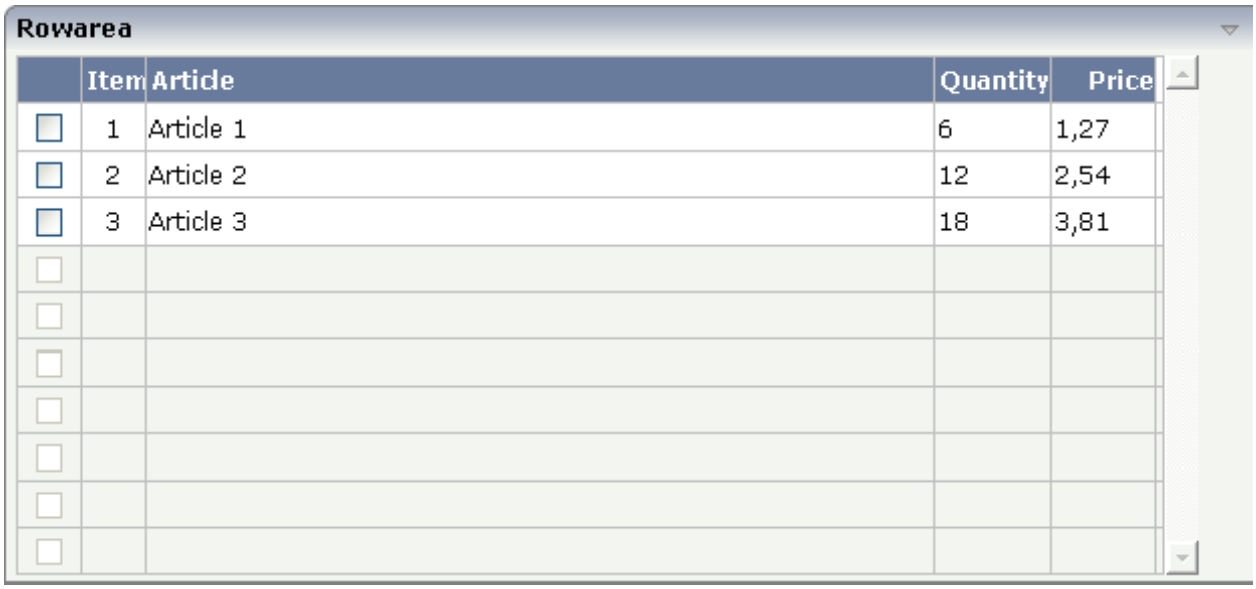
If using input controls (FIELD, CHECKBOX, COMBOBOX) inside a grid, then there are two special events that may be passed to your application.

- `FWDTABKEYMETHOD` of ROWTABLEAREA2: this is the method that is called when the user presses the TAB key within the rightmost field of the grid.
- `BWDTABKEYMETHOD` of ROWTABLEAREA2: this is the method that is called when the user presses SHIFT+TAB on the leftmost control of the grid.

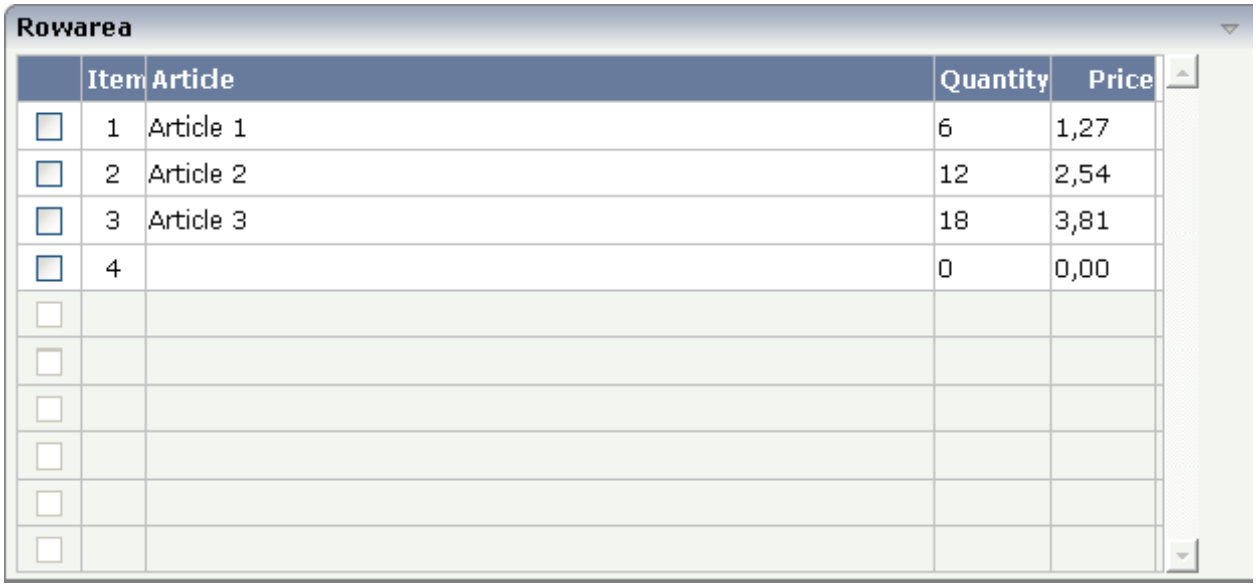
You can use these events for various purposes:

- You may create a new item below the existing one when the user leaves the rightmost field.
- You may want to trigger the scrolling of the grid if the user tabs through the last right field.

In the following example, every time the user leaves the rightmost field of the grid, a new item is created:



When the user now presses TAB in the last field, the screen will look as follows:



The XML layout is:

```
...
...
...
<rowtablearea2 griddataprop="lines" rowcount="10" ↩
fwdtabkeymethod="endofLineProcessing">
    <tr>
       <label name=" " width="30" asheadline="true" labelstyle="text-align:center">
        </label>
        <label name="Item" width="30" asheadline="true">
        </label>
        <label name="Article" width="400" asheadline="true">
        </label>
        <label name="Quantity" asheadline="true" labelstyle="text-align:right">
        </label>
        <label name="Price" width="50" asheadline="true" ↩
labelstyle="text-align:right">
        </label>
    </tr>
    <repeat>
        <str valueprop="selected">
            <checkbox valueprop="selected" flush="screen" width="30">
            </checkbox>
            <textout valueprop="itemNumber" width="30" ↩
textoutstyle="text-align:center">
            </textout>
            <field valueprop="article" width="100%" noborder="true"
                   transparentbackground="true">
            </field>
            <field valueprop="quantity" width="50" noborder="true"
                   transparentbackground="true">
            </field>
```

```
            <field valueprop="price" width="50" noborder="true"
                  transparentbackground="true" datatype="float" decimaldigits="2">
            </field>
            <hdist>
            </hdist>
        </str>
    </repeat>
</rowtablearea2>
...
...
```

Be aware that the method that is associated with the “tab” event is called in the item object in which the “tab” event was thrown.

The server-side processing looks as follows:

```
...
...
public class rowtab2_specEventsAdapter
    extends Adapter
{
    // class >LinesItem<
    public class LinesItem
    {
        public LinesItem(int itemNumber)
        {
             m_itemNumber = itemNumber;
             m_focussedItemNumber = itemNumber;
        }

        public String getArticleStatus()
        {
            if (m_itemNumber == m_focussedItemNumber)
                return "FOCUS";
            else
                return "EDIT";
        }

        int m_itemNumber;
        public int getItemNumber() { return m_itemNumber; }
        public void setItemNumber(int value) { m_itemNumber = value; }

        int m_focussedItemNumber;
        public int getfocussedItemNumber() { return m_itemNumber; }
        public void setfocussedItemNumber(int value) { m_itemNumber = value; }

        // property >article<
        String m_article;
        public String getArticle() { return m_article; }

        // property >price<
```

```
        double m_price;
        public double getPrice() { return m_price; }

        // property >quantity<
        int m_quantity;
        public int getQuantity() { return m_quantity; }

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

        public void endOfLineProcessing()
        {
            if (m_itemNumber == m_lines.size())
            {
                LinesItem item = new LinesItem(m_lines.size() + 1);
                m_lines.add(item);
                m_lines.displayItem(item);
            }
        }
    }
...
...
```

Inside the method `endOfLineProcessing()` that is defined on item level, the new item is created. In addition, you see that there is a certain focus management behind the field representing the article: the focus management is used in a way that the focus is directly set into the article field when a new item is created.

The full XML and code is available inside the project cisdemos.

ROWTABLEAREA2 Properties

Basic			
griddataprop	Name of adapter property representing the grid on server side. Must be of type "GRIDCollection". The whole grid is represented by the GRIDCollection-object, each individual row of the grid is represented by one item inside the collection. If using the control for building trees (TREENODE-control inside the grid's	Obligatory	

	items) then use "TREECollection" on server side.		
rowcount	Number of rows that are rendered inside the control. There are two ways of using this property - dependent on whether you in addition define the HEIGHT property: If you do NOT define the HEIGHT property then the control is rendered with exactly the number of rows that are defined as ROWCOUNT value. If a HEIGHT value is defined in addition (e.g. as percentage value "100%") then the number of rows depend on the actual height of the control. The ROWCOUNT value in this case indicates the maximum number of rows that are picked from the server. You should define this value in a way so that it is not too low - otherwise your grid will not be fully filled. On the other hand it should not be defined too high ("100") because this causes more communication traffic and more rendering effort inside the browser.	Optional	
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may	Optional	100 150 200 250 300 250 400 50% 100%

	itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		
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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%
firstrowcolwidths	If set to "true" then the grid is sized according to its first row. This first row typically is a header-TR-row in which GRIDCOLHEADER controls are used as column headers for the subsequent rows. Default is "false", i.e. the grid is sized according to its "whole content". Please note: when using the GRIDCOLHEADER control within the header-TR-row this property must be set to "true" - otherwise column resizing (by drag and drop) does not work correctly.	Sometimes obligatory	true false
onloadbehaviour	DEPRECATED: Loading behaviour of the items into the client. "block" (=default). Only "block" is supported´. All other settings are automatically mapped to "block".	Optional	block collection collectionorblock

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
withborder	If set to "false" then no thin border is drawn around the controls that are contained in the grid. Default is "true".	Optional	true false
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "hidden".	Optional	auto scroll hidden
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "scroll".	Optional	auto scroll hidden
firstrowcolwidths	(already explained above)		
frozencolumnsleft	Number of columns, which should be horizontally frozen to the left.	Optional	1 2 3 int-value
touchpadinput	If set to "true" then touch screen icons for scrolling are displayed in addition. Default is "false".	Optional	true false
requiredheight	Minimum height of the control in pixels. Use this property to ensure a minimum height if the overall control's height is a percentage of the available space - i.e. if	Optional	1 2 3

	value of property HEIGHT is a percentage (e.g. 100%). Please note:You must not use FIXLAYOUT at the surrounding row container (ITR and ROWAREA). Otherwise: if the available space is less than the required height the end of the control is just cut off.		int-value
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
darkbackground	Normally the background is in light colour but the CIS style sheets also have a dark(er) grey colour to be used. If DARKBACKGROUND is set to true then the darker background colour is chosen. This property typically is used to integrate light coloured controls into darker container areas.	Optional	true false
invisiblemodeincompletelastrow	If set to "invisible" an incomplete last row is not shown.	Optional	invisible visible
withsliderfreeze	Setting this to "true" prevents unwanted slider jumps while scrolling up/down in a grid with a huge number of lines (for example 20000).	Optional	true false

Features			
clipboardaccess	If switched to true then the content of the grid can be selected and exported into the client's clipboard.	Optional	true false
withblockscrolling	If switched to "true" then the grid will show small scroll icons by which the user can scroll the grid's content. Scrolling typically is done by using the grid's scrollbar - the scroll icons that are switched on by this property are an additional possibility to scroll.	Optional	true false
withcopytoclipboard	If set to TRUE a corresponding icon to copy all grid items to the clipboard is added.	Optional	true false
copytoclipboardtitle	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	
formatcopytoclipboard	formatcopytoclipboard	Optional	true false
withselectedtoclipboard	If set to TRUE a corresponding icon to copy the selected grid items to the clipboard is added.	Optional	true false
selectedtoclipboardtitle	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	
withexporttofile	If set to TRUE a corresponding icon to copy all grid items to a csv file is added.	Optional	true false
exporttofiletitle	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	

withselectedtofile	If set to TRUE a corresponding icon to copy the selected grid items to a csv file is added.	Optional	true false
selectedtofiletitle	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	
withexportheaders	If set to TRUE a corresponding icon for toggling the export of headers is added.	Optional	true false
exportheaderstitle	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	
exportheadersprop	Name of the adapter property that presents the value for exporting headers at runtime. At runtime "true" switches on the export of headers.	Optional	
Binding			
oncontextmenumethod	Name of adapter method that is called when the user presses right mouse button into the grid - but not on an existing row (then the row item object is responsible for handling the right mouse button) but on "empty area" of the grid.	Optional	
fwdtabkeymethod	Name of an adapter method that is called if the user presses the TAB key within the very last cell of the grid (last cell within the last line). Use property FWDTABKEYFILTER to associate this call with a grid column.	Optional	
fwdtabkeyfilter	By default the FWDTABKEYMETHOD is called if the user presses the TAB key within the veryfirst cell of the grid. Input the name of a cell's VALUEPROP to associate the method call with any other column.	Optional	
bwdtabkeymethod	Name of an adapter method that is called if the user presses SHIFT and TAB keys	Optional	

	within the first cell of a grid line. Use property BWDTABKEYFILTER to associate this call with a cell of choice.		
bwdtabkeyfilter	By default the BWDTABKEYMETHOD is called if the user presses the SHIFT and TAB keys within the very first cell of the grid. Input the name of a cell's VALUEPROP to associate the method call with any other column.	Optional	
frozencolumnsleftprop	Name of the adapter property which dynamically provides the number of columns to be be frozen to the left.	Optional	
Hot Keys			
hotkeys	Semicolon separated list of hot keys. A hotkey consists of a list of keys and a method name. Separate the keys by "-" and the method name again with a semicolon Example: ctrl-alt-65;onCtrlAltA;13;onEnter ...defines two hot keys. Method onCtrlAltA is invoked if the user presses Ctrl-Alt-A. Method "onEnter" is called if the user presses the ENTER key. Use the popup help within the Layout Painter to input hot keys.	Optional	

STR Properties

STR (selectable table row) is a normal table row (TR) that highlights its background depending on an adapter property.

Basic			
valueprop	Name of the adapter property that defines if the row is selected (value "true") or not selected ("false").	Obligatory	
withalterbackground	Flag that indicates if the grid line shows alternating background color (like rows within a textgrids). Default is false. Please note: controls inside the row must have transparent background. In case of the FIELD control simply set property TRANSPARENTBACKGROUND to true.	Optional	true false

showifempty	Flag that indicates if an unused row is visible. Example: if set to false a grid with rowcount ten and a server side collection size of seven will hide the three remaining rows. Default is false.	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)		
onclickmethod	Name of the method inside the row item class that is called if the user clicks.	Optional	
alwaysonclick	If no onclickmethod is defined, this property is ignored. If an onclickmethod is defined and this property is set to false, then the onclickmethod is only called if the clicked control does call a method itself. Example: A click on a button will only call the method defined in the button. Default is true, which means both methods will be called.	Optional	true false
ondblclickmethod	Name of the method that is called when the user double clicks.	Optional	
contextmenumethod	Name of the method on adapter level that is called when the user presses the right mouse button in an empty area.	Optional	
proprefprop	Name of the property inside the row item class that is called if the user clicks a FIELD control. The VALUEPROP of the clicked field control will passed.	Optional	
strstyleprop	Name of adapter property that dynamically provides explicit style information for the control.	Optional	
backgroundcolorprop	Name of the adapter property that dynamically sets the background color for the control.	Optional	

In the above **example**, the selection itself is done by a CHECKBOX. Both CHECKBOX and STR definitions are bound to the same Boolean value property (`selected`). Because of the `flush` definition inside the CHECKBOX, the table row is highlighted immediately after clicking the check box.

Inside the REPEAT definition, you can use also normal table rows (TR instead of STR). You cannot use ITR table rows to form a well-structured grid, because all columns have to be synchronized in their width.

It is recommended to use STR rows. The reasons are:

- The STR row refers to a property representing its selection status (property `valueprop`). If this property is not available, the STR row automatically deactivates its contained controls. This means: if the STR row is not represented by a corresponding data object on the server side (because the grid contains more rows than are made available by the grid collection), then all controls of the STR row are automatically deactivated.

- Special grid functions like up/down cursor navigation and cut/paste operations with the right mouse button are only available with the STR row, not with TR.

62 COLINFOS Control - Show and Hide Single Columns

COLINFOS is a container control that allows to dynamically show and hide single columns in a ROWTABLEAREA2 control. The COLINFOS container can only contain COLINFO controls. Each COLINFO control is responsible for one column.

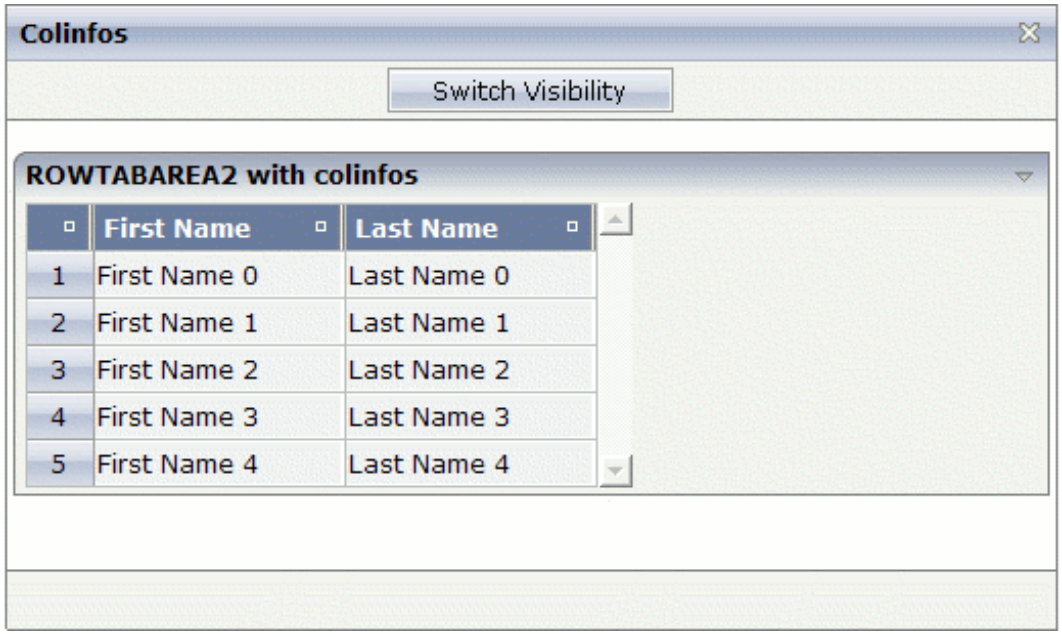
Notes:

1. This feature can only be used with Internet Explorer. It cannot be used with Mozilla Firefox or Netscape since these browsers do not support the `COLGROUP` and `COL` elements as defined with the HTML 4.01 specification.
2. The COLINFOS control is deprecated. It is recommended that you use the `visibleprop` property of the **GRIDCOLHEADER** control instead.

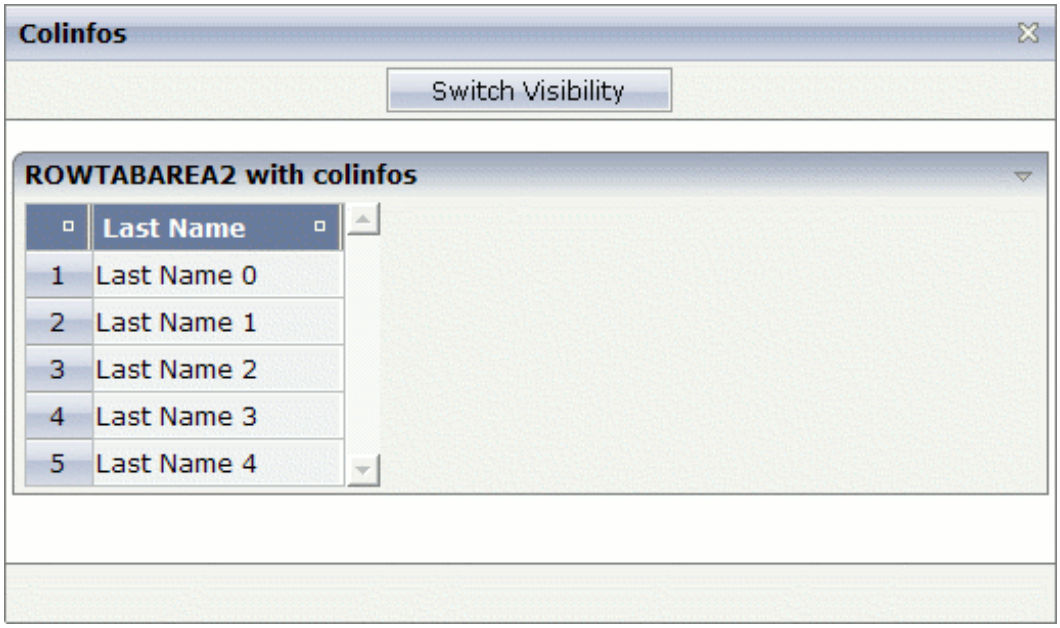
Example

The table in this example has three columns: the first column contains a SELECTOR control for the numbers, the second column contains a FIELD control for the first names, and the third column contains a FIELD control for the last names.

The second column (first name) can be visible.



You can also hide the second column (first name).



The XML layout definition is:

```
<pagebody>
  <rowarea name="ROWTABAREA2 with colinfos">
  <rowtablearea2 griddataprop="lines" rowcount="5">
    <colinfos>
    <colinfo></colinfo>
    <colinfo visibleprop="col2"></colinfo>
    </colinfos>
    <tr>
    <gridcolheader name=" " width="30">
    </gridcolheader>
    <gridcolheader name="First Name" width="120">
    </gridcolheader>
    <gridcolheader name="Last Name" width="120">
    </gridcolheader>
    </tr>
    <repeat>
    <str valueprop="selected">
      <selector valueprop="selected" width="30" singleselect="true">
      </selector>
      <field valueprop="fname" width="120" displayonly="true"
         noborder="true">
      </field>
      <field valueprop="lname" width="120" displayonly="true"
         noborder="true">
      </field>
    </str>
    </repeat>
  </rowtablearea2>
```

```
  </rowarea>
</pagebody>
```

A COLINFO control is required for each column that is to be shown/hidden dynamically. The responsibility for the columns goes from left to right. Thus, the first COLINFO control applies to the first column, the second COLINFO control applies to the second column, and so on.

In the above example, two COLINFO controls are used even though only the second column is to be shown/hidden. The first COLINFO control is required since the columns are calculated from left to right. Since the first COLINFO control does not have a `visibleprop` property, this column is always visible.

If you would omit the first COLINFO control, the `visibleprop` property of the remaining COLINFO control would be applied to the first column containing the numbers (and not to the second column containing the first name).

A third COLINFO control is not required in this example, since you do not want to show/hide the third column containing the last name.

The Java code of the adapter is:

```
public class colinfosAdapter
extends Adapter
{
  // property >col2<
  boolean m_col2 = true;
  public boolean getCol2() { return m_col2; }
  public void setCol2(boolean value) { m_col2 = value; }

  // class >LinesItem<
  public class LinesItem
  {
    // property >fname<
    String m_fname;
    public String getFname() { return m_fname; }
    public void setFname(String value) { m_fname = value; }

    // property >lname<
    String m_lname;
    public String getLname() { return m_lname; }
    public void setLname(String value) { m_lname = 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()
  {
    // generate some content for the displayed lines ...
    LinesItem item;
    for (int i = 0; i < 5; i++)
    {
      item = new LinesItem();
      item.m_fname = "First Name " +i;
      item.m_lname = "Last Name "+i;
      m_lines.add(item);
    }
  }

  public void onSwitch()
  {
  // switch visibility of the second column
    m_col2 = !m_col2;
  }
}
```

COLINFOS Properties

Basic			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

COLINFO Properties

Basic			
visibleprop	Name of property that tells if the corresponding column that is associated with the colinfo-control is displayed or not. Property must be of type "boolean".	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

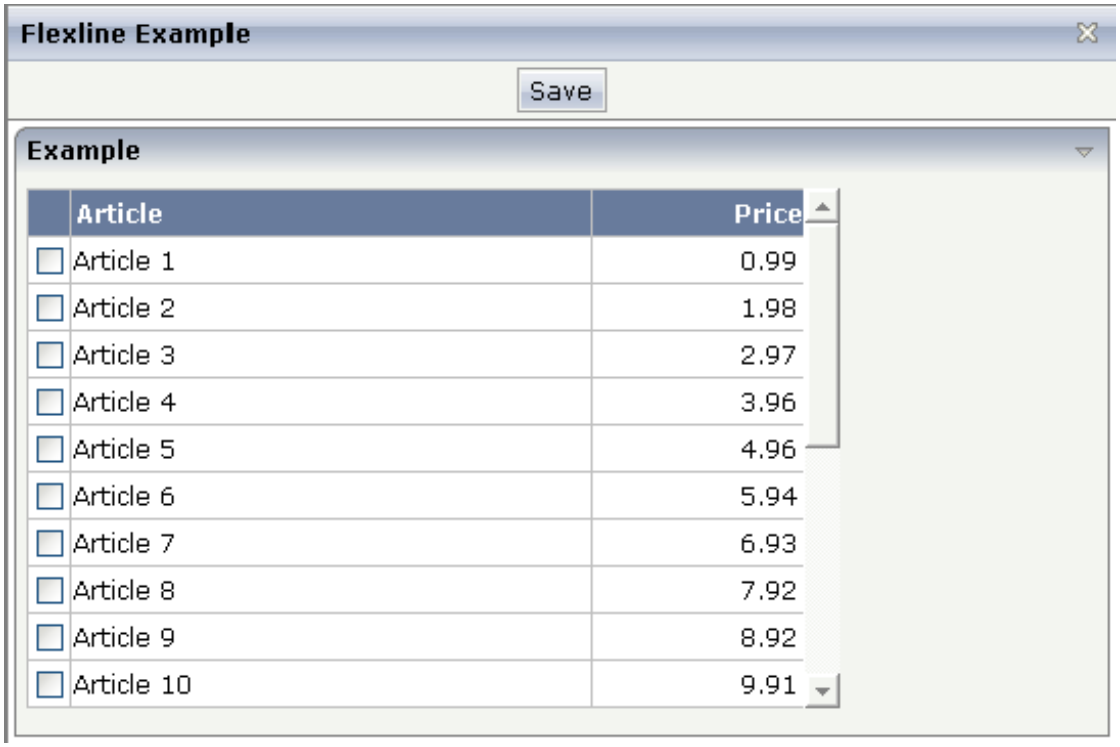
63 FLEXLINE - Flexible Columns in Control Grids

In a **previous** example, the grid was completely defined as part of the layout definition: the sequence of columns was internally defined by defining the controls that are part of an STR row.

The FLEXLINE control offers the option to define the columns of a grid dynamically at runtime. That is: the application decides at runtime which column controls to use with which properties. Consequently, you can build a control grid with some system configuration in mind, in which the layout of control grids is customized.

Example

Have a look at the following example:



The grid looks like a normal ROWTABLEAREA2 grid, but it is built in a more dynamic way.

The XML layout definition is:

```
<page model="flexline_01Adapter">
    <titlebar name="Flexline Example">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
        <rowarea name="Example">
            <vdist height="5">
            </vdist>
            <rowtablearea2 griddataprop="lines" rowcount="10" width="395" ↩
withborder="true">
                <tr>
                    <label name=" " asheadline="true">
                    </label>
                    <flexline infoprop="headline">
                    </flexline>
                </tr>
                <repeat>
                    <str valueprop="selected">
                        <checkbox valueprop="selected" flush="screen" width="30">
                        </checkbox>
                        <flexline infoprop="/rowline">
                        </flexline>
                        <hdist width="100%">
                        </hdist>
                    </str>
                </repeat>
            </rowtablearea2>
            <vdist height="10">
            </vdist>
        </rowarea>
        <vdist height="5">
        </vdist>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You see that there are two FLEXLINE control definitions inside the ROWTABLEAREA2 definition:

- One definition represents the headline of the grid.
- The other definition is part of each row's content.

Each definition points to a property that passes the configuration at runtime. Within the second definition, you may see something which is new for you: the VALUEPROP references to a property `/rowline`. The "/" character at the beginning indicates that this property is always picked from the adapter - and not from the object representing the row item.

This is the Java code on the server side:

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

import java.math.BigDecimal;

import com.softwareag.cis.file.CSVManager;
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.FLEXLINEInfo;
import com.softwareag.cis.server.util.GRIDCollection;

public class flexline_01Adapter
    extends Adapter
{
    // class >LinesItem<
    public class LinesItem
    {
        // property >selected<
        boolean m_selected;
        public boolean getSelected() { return m_selected; }
        public void setSelected(boolean value) { m_selected = value; }

        String m_article;
        BigDecimal m_price;

        public void remove()
        {
            m_lines.remove(this);
        }

        public String getArticle() { return m_article; }
        public void setArticle(String article) { m_article = article; }
        public BigDecimal getPrice() { return m_price; }
        public void setPrice(BigDecimal price)
        {
            m_price = price.setScale(2,BigDecimal.ROUND_UP);
        }
    }

    // property >headline<
    FLEXLINEInfo m_headline = new FLEXLINEInfo();
    public FLEXLINEInfo getHeadline() { return m_headline; }

    // property >rowline<
    FLEXLINEInfo m_rowline = new FLEXLINEInfo();
    public FLEXLINEInfo getRowline() { return m_rowline; }

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

    /** initialisation - called when creating this instance*/
    public void init()
    {
```

```
        // configure controls in headline
        m_headline.addLabel(this,CSVManager.encodeString(new String[]
        {
                "name","Article",
                "width","250",
                "asheadline","true"
        }));
        m_headline.addLabel(this,CSVManager.encodeString(new String[]
        {
                "name","Price",
                "width","100",
                "textalign","right",
                "asheadline","true"
        }));
        // configure controls in row
        m_rowline.addField(this,CSVManager.encodeString(new String[]
        {
                "valueprop","article",
                "width","250",
                "flush","server",
                "noborder","true",
                "transparentbackground","true"
        }));
        m_rowline.addField(this,CSVManager.encodeString(new String[]
        {
                "valueprop","price",
                "width","100",
                "textalign","right",
                "noborder","true",
                "transparentbackground","true"
        }));
        // create lines
        for (int i=1; i<=20; i++)
        {
            LinesItem li = new LinesItem();
            li.setArticle("Article " + i);
            li.setPrice(new BigDecimal(i*0.99));
            m_lines.add(li);
        }
    }
}
```

For each FLEXLINE control, there is a `FLEXLINEInfo` property. The properties are initialized during the `init()` phase of the adapter. Of course, you can also change the `FLEXLINEInfo` configuration later: there is a corresponding `clear()` method for doing so.

Inside the `FLEXLINEInfo` class, there is a Java interface with which you can add:

- labels
- check boxes

- buttons
- combo boxes

There is a method for each object. As part of the method, you always pass the owner (i.e. the current model) and the configuration of the control.

The configuration is passed as a comma-separated string that is built using the `CSVManager` class. You could also directly write the CSV string (`valueprop;price;width;100;noborder;true`) but then have to be careful to replace every “real” semicolon character with "\;". You can use and combine any properties that are available for the controls. This means there is no difference in managing controls that you flexibly add and controls that are defined in a fixed way inside a layout definition.

All the other processing around the FLEXLINE management is the same as you know it from layouts that are defined in a fixed way.

Note: Application Designer now provides a FLEXGRID control. While FLEXLINE still is supported (and necessary), FLEXGRID offers a simpler API to build dynamically controlled control grids. Basically, FLEXGRID is a combination of ROWTABLEAREA2, FLEXLINE and GRIDCOLHEADER. See the description of the **FLEXGRID** control.

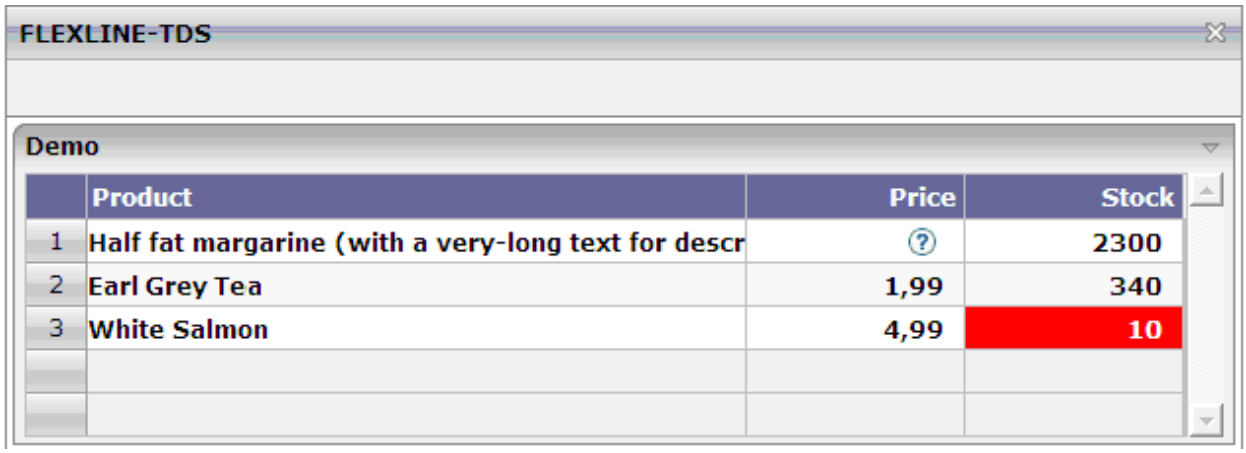
FLEXLINE Properties

Basic			
infoprop	Name of the adapter property that provides the server side information for this control. The adapter property must be of type "FLEXLINEInfo". Inside the property the sequence of controls is defined.	Obligatory	
withborder	Flag that indicates if a border is drawn between the controls that are rendered inside the FLEXLINE control. Default is "false", i.e. no border is drawn.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Increasing the Performance

In the above example, the grid was filled by adding FIELD controls into a FLEXLINE control. In larger grids with a high number of columns and rows, you may consider displaying your data within plain <TD> elements. Thus, the grid becomes more lightweight and the performance increases significantly. You can output images and use your own cell style. Direct cell input, however, is not possible.

In the following example, the grid looks like a ROWTABLEAREA2 grid, but it is built in a more dynamic way.



The XML layout definition is:

```
<page model="FlexlineTDSAdapter">
    <titlebar name="FLEXLINE-TDS">
    </titlebar>
    <header withdistance="false">
    </header>
    <pagebody>
        <rowarea name="Demo">
            <rowtablearea2 griddataprop="lines" rowcount="5" width="100%" ↩
firstrowcolwidths="true">
                <tr>
                    <label name=" " width="30" asheadline="true">
                    </label>
                    <label name="Product" width="60%" asheadline="true">
                    </label>
                  <label name="Price&amp;amp;nbsp;" width="20%" asheadline="true" ↩
textalign="right">
                    </label>
                  <label name="Stock&amp;amp;nbsp;" width="20%" asheadline="true" ↩
textalign="right">
                    </label>
```

```
                </tr>
                <repeat>
                    <str valueprop="selected" withalterbackground="true">
                        <selector valueprop="selected">
                        </selector>
                        <flexline infoprop="/contentFL">
                        </flexline>
                    </str>
                </repeat>
            </rowtablearea2>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You see that there is one FLEXLINE control definition inside the ROWTABLEAREA2 definition. This definition points to a property that passes the configuration at runtime. Within the definition, the `infoprop` references to a property "/contentFL". The slash (/) at the beginning indicates that this property is always picked from the adapter (it is not picked up from the object representing the row item).

This is the Java code on the server side:

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

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class FlexlineTDSAdapter extends Adapter
{
  // class >LinesItem<
  public class LinesItem
  {
    // property >selected<
    boolean m_selected;

    public boolean getSelected()
    {
      return m_selected;
    }

    public void setSelected(boolean value)
    {
      m_selected = value;
    }

    // property >tdsValue<
    String m_tdsValue;
```

```
  public String getTdsValue()
  {
    return m_tdsValue;
  }

  // property >tdsColors<
  String m_tdsColor;

  public String getTdsColor()
  {
    return m_tdsColor;
  }

  // property >tdsBGColors<
  String m_tdsBGColor;

  public String getTdsBGColor()
  {
    return m_tdsBGColor;
  }

  // property >tdsAlign<
  String m_tdsAlign;

  public String getTdsAlign()
  {
    return m_tdsAlign;
  }

  // property >imageURL<
  String m_imageURL;

  public String getImageURL()
  {
    return m_imageURL;
  }
}

// property >contentFL<
FLEXLINEInfo m_contentFL = new FLEXLINEInfo();

public FLEXLINEInfo getContentFL()
{
  return m_contentFL;
}

// property >lines<
GRIDCollection m_lines = new GRIDCollection();

public GRIDCollection getLines()
{
```

```
    return m_lines;
  }

  /** initialization - called when creating this instance */
  public void init()
  {
    m_contentFL.addTds(this,
           "colcount;5;" +
           "valueprop;tdsValue;" +
           "fgcolorprop;tdsColor;" +
           "bgcolorprop;tdsBGColor;" +
           "alignprop;tdsAlign;" +
           "imageprop;imageURL;" +
           "font-weight;bold");

    LinesItem item = new LinesItem();
    item.m_tdsValue = "Half fat margarine (with a very-long text for ↩
description);0,99;2300";
    item.m_tdsColor = "#000000;#000000;#000000";
    item.m_tdsBGColor = ";;";
    item.m_imageURL = ";../HTMLBasedGUI/images/helpiconblue.gif;";
    item.m_tdsAlign = "left;right;right";
    m_lines.add(item);

    item = new LinesItem();
    item.m_tdsValue = "Earl Grey Tea;1,99;340";
    item.m_tdsColor = "#000000;#000000;#000000";
    item.m_tdsBGColor = ";;";
    item.m_imageURL = ";;";
    item.m_tdsAlign = "left;right;right";
    m_lines.add(item);

    item = new LinesItem();
    item.m_tdsValue = "White Salmon;4,99;10";
    item.m_tdsColor = "#000000;#000000;#FFFFFF";
    item.m_tdsBGColor = ";;#FF0000";
    item.m_imageURL = ";;";
    item.m_tdsAlign = "left;right;right";
    m_lines.add(item);

  }
}
```

For each FLEXLINE control, there is a `FLEXLINEInfo` property. The properties are initialized during the `init()` phase of the adapter. You can also change the `FLEXLINEInfo` configuration later: there is a corresponding `clear()` method for doing so.

Inside the `FLEXLINEInfo` class, there is a Java interface with which you can add the following:

```
addTds(Adapter owner, String properties);
```

Semicolons are used to separate the values in the property list. The properties may contain the following values:

colcount
The number of columns that have to be provided for the contents.

valueprop
The property name of the row item class that returns a list of column values which are separated by semicolons.

fgcolorprop
The property name of the row item class that returns a list of foreground color values which are separated by semicolons.

bgcolorprop
The property name of the row item class that returns a list of background color values which are separated by semicolons.

alignprop
The property name of the row item class that returns a list of cell alignment values which are separated by semicolons. Possible values: left, center, right.

imageprop
The property name of the row item class that returns a list of image URLs which are separated by semicolons. A single TD contains either text (see `valueprop`) or an image. If you provide an image URL and text for same cell, the text is suppressed.

font-weight
The weight of the font.

A width is not passed. It is assumed that the width is defined by the environment (for example, by a ROWTABLEAREA2 control where the columns have a fixed size).

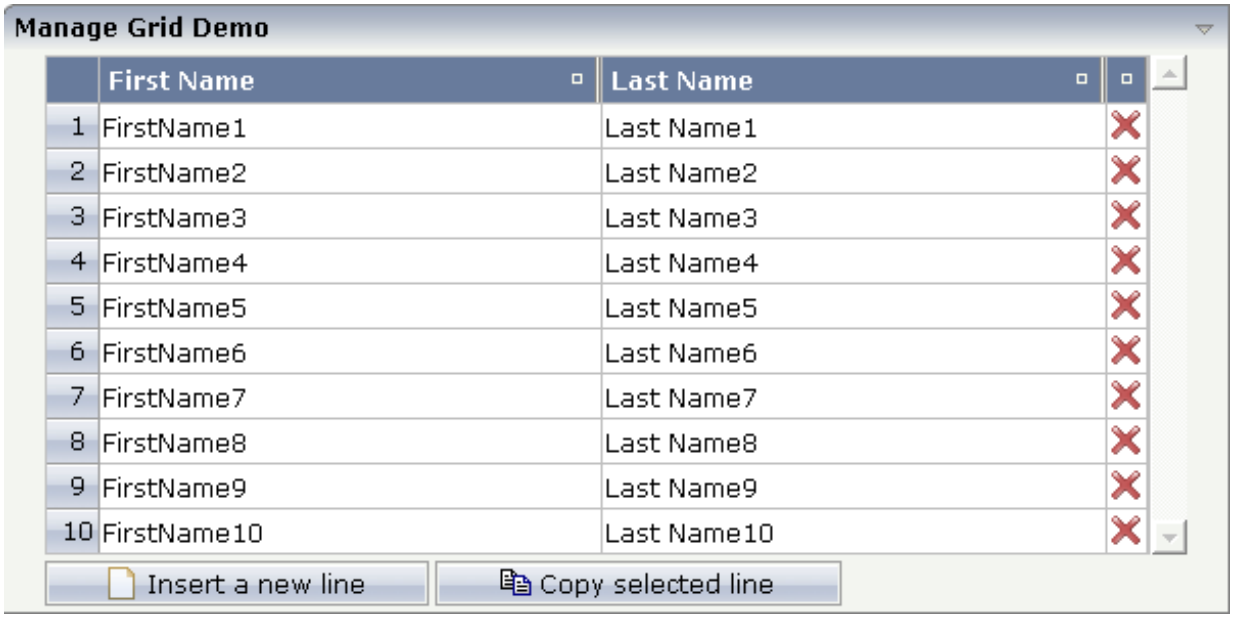
Note: It is currently only possible to add exactly one `Tds` control to one FLEXLINE control.

64 MGDGRID - Managing the Grid

The MGDGRID control is an extension of the **ROWTABLEAREA2** control. It allows to insert, copy and delete rows of the grid without the need of any corresponding Java coding.

See also ***STR Properties*** which are described with the ROWTABLEAREA2 control.

Example



There is a grid that contains a header row and 10 lines. Each line contains two fields and a “delete row” control.

Each of the function controls (insert, copy, delete) can be added at the top of the MGDGRID, below the MGDGRID or within the lines of the MGDGRID.

Look at the corresponding layout definition:

```
<rowarea name="Manage Grid Demo">
  <mgdgrid griddataprop="mglines" rowcount="10" width="100%" firstrowcolwidths="true">
    <tr>
      <label name="  " width="25" asheadline="true">
      </label>
      <gridcolheader name="First Name" width="50%">
      </gridcolheader>
      <gridcolheader name="Last Name" width="50%" >
      </gridcolheader>
      <gridcolheader width="20">
      </gridcolheader>
      <hdist></hdist>
    </tr>
```

```
    <repeat>
      <str valueprop="selected" showifempty="true">
        <selector valueprop="selected" singleselect="true">
        </selector>
        <field valueprop="fname" width="100%">
        </field>
        <field valueprop="lname" width="100%">
        </field>
        <rowdelete>
        </rowdelete>
      </str>
    </repeat>
    <mgdfunctions>
      <rowinsert title="Insert a new line">
      </rowinsert>
      <rowcopy title="Copy selected line">
      </rowcopy>
    </mgdfunctions>
  </mgdgrid>
</rowarea>
```

This is the corresponding adapter code:

```
public class ManageGridDemoAdapter
  extends Adapter
{
    // property >mglines<
    MGDGRIDCollection m_mglines = new MGDGRIDCollection(MGDItem.class);
    public GRIDCollection getMglines() { return m_mglines; }
}
```

The constructor of the `MGDGRIDCollection` class needs an “item class”. This “item class” cannot be a sub/inner class this time; therefore two separate Java files are required for the MGDGRID.

This is the Java code of the corresponding `MGDItem` class:

```
public class MGDItem
{
    // property >fname<
    String m_fname;
    public String getFname() { return m_fname; }
    public void setFname(String value) { m_fname = value; }

    // property >lname<
    String m_lname;
    public String getLname() { return m_lname; }
    public void setLname(String value) { m_lname = value; }

    // property >selected<
    boolean m_selected;
    public boolean getSelected() { return m_selected; }
```

```
    public void setSelected(boolean value) { m_selected = value; }
}
```

With the MGDGRID control, there is no need of further Java coding. It handles insert, copy and delete events itself, but the developer has to take care of the corresponding “item class”.

The MGDGRID control is an extension to the ROWTABLEAREA2 control. See the description of the **ROWTABLEAREA2** control for further information.

MGDGRID Properties

Basic			
griddataprop	Name of adapter property representing the grid on server side. Must be of type "GRIDCollection". The whole grid is represented by the GRIDCollection-object, each individual row of the grid is represented by one item inside the collection. If using the control for building trees (TREENODE-control inside the grid's items) then use "TREECollection" on server side.	Obligatory	
rowcount	Number of rows that are rendered inside the control. There are two ways of using this property - dependent on whether you in addition define the HEIGHT property: If you do NOT define the HEIGHT property then the control is rendered with exactly the number of rows that are defined as ROWCOUNT value. If a HEIGHT value is defined in addition (e.g. as percentage value "100%") then the number of rows depend on the actual height of the control. The ROWCOUNT value in this case indicates the maximum number of rows that are picked from the server. You should define this value in a way so that it is not too low - otherwise your grid will not be fully filled. On the other hand it should not be defined too high ("100") because this causes more communication traffic and more rendering effort inside the browser.	Optional	
height	Height of the control.	Optional	100

	There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		150 200 250 300 250 400 50% 100%
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 percentage 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.	Sometimes obligatory	100 120 140 160 180 200 50% 100%
firstrowcolwidths	If set to "true" then the grid is sized according to its first row. This first row typically is a header-TR-row in which GRIDCOLHEADER controls are used as column headers for the subsequent rows.	Sometimes obligatory	true false

	Default is "false", i.e. the grid is sized according to its "whole content". Please note: when using the GRIDCOLHEADER control within the header-TR-row this property must be set to "true" - otherwise column resizing (by drag and drop) does not work correctly.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
withborder	If set to "false" then no thin border is drawn around the controls that are contained in the grid. Default is "true".	Optional	true false
hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "hidden".	Optional	auto scroll hidden
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "scroll".	Optional	auto scroll hidden
firstrowcolwidths	(already explained above)		
clipboardaccess	If switched to true then the content of the grid can be selected and exported into the client's clipboard.	Optional	true false
withblockscrolling	If switched to "true" then the grid will show small scroll icons by which the user can scroll the grid's content. Scrolling typically is done by using the grid's scrollbar - the scroll icons that are switched on by this property are an additional possibility to scroll.	Optional	true false
touchpadinput	If set to "true" then touch screen icons for scrolling are displayed in addition. Default is "false".	Optional	true false

requiredheight	Minimum height of the control in pixels. Use this property to ensure a minimum height if the overall control's height is a percentage of the available space - i.e. if value of property HEIGHT is a percentage (e.g. 100%). Please note:You must not use FIXLAYOUT at the surrounding row container (ITR and ROWAREA). Otherwise: if the available space is less than the required height the end of the control is just cut off.	Optional	1 2 3 int-value
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Binding			
oncontextmenumethod	Name of adapter method that is called when the user presses right mouse button into the grid - but not on an existing row (then the row item object is responsible for handling the right mouse button) but on "empty area" of the grid.	Optional	
fwdtabkeymethod	Name of an adapter method that is called if the user presses the TAB key within the very last cell of the grid (last cell within the last line). Use property FWDTABKEYFILTER to associate this call with a grid column.	Optional	
fwdtabkeyfilter	By default the FWDTABKEYMETHOD is called if the user presses the TAB key within the veryfirst cell of the grid. Input the name of a cell's VALUEPROP to associate the method call with any other column.	Optional	
bwdtabkeymethod	Name of an adapter method that is called if the user presses SHIFT and TAB keys within the first	Optional	

	cell of a grid line. Use property BWDTABKEYFILTER to associate this call with a cell of choice.		
bwdtabkeyfilter	By default the BWDTABKEYMETHOD is called if the user presses the SHIFT and TAB keys within the very first cell of the grid. Input the name of a cell's VALUEPROP to associate the method call with any other column.	Optional	
Hot Keys			
hotkeys	Semicolon separated list of hot keys. A hotkey consists of a list of keys and a method name. Separate the keys by "-" and the method name again with a semicolon Example: ctrl-alt-65;onCtrlAltA;13;onEnter ...defines two hot keys. Method onCtrlAltA is invoked if the user presses Ctrl-Alt-A. Method "onEnter" is called if the user presses the ENTER key. Use the popup help within the Layout Painter to input hot keys.	Optional	

ROWINSERT Properties

Basic			
image	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
Online Help			

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 language management - representing the tooltip text that is used for the control.	Optional	

ROWCOPY Properties

Basic			
image	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

ROWDELETE Properties

Basic			
image	URL that points to the image that is shown as icon. The URL either is an absolute URL or a relative URL. If using a relative URL then be aware of that the generated page is located directly inside your project's directory. Example: "images/icon.gif" points to an icon in an images-folder that is parallel to the page itself. "../HTMLBasedGUI/images/new.gif" point to a URL that is located inside a different project.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Binding			
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
Online Help			
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 language management - representing the tooltip text that is used for the control.	Optional	

65 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.

Flexible Column Sizing

Let us have a look on the following grid definition:

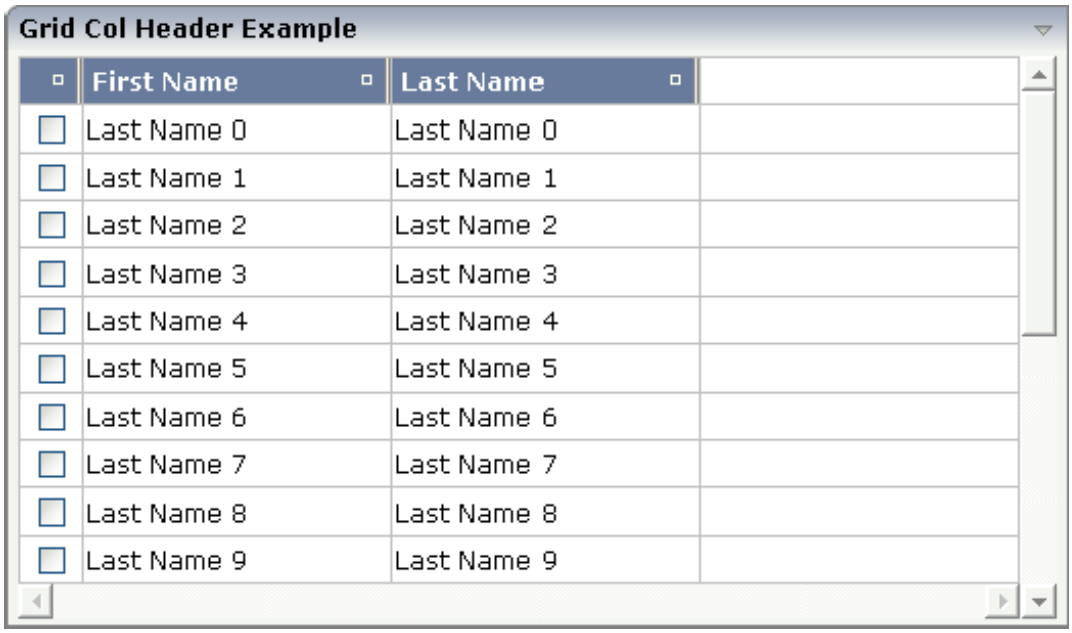
```
<rowarea name="Grid Col Header Example">
   <rowtablearea2 griddataprop="lines" rowcount="10" width="100%" withborder="true"
                  hscroll="true" firstrowcolwidths="true">
       <tr>
           <gridcolheader name=" " width="30">
           </gridcolheader>
           <gridcolheader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

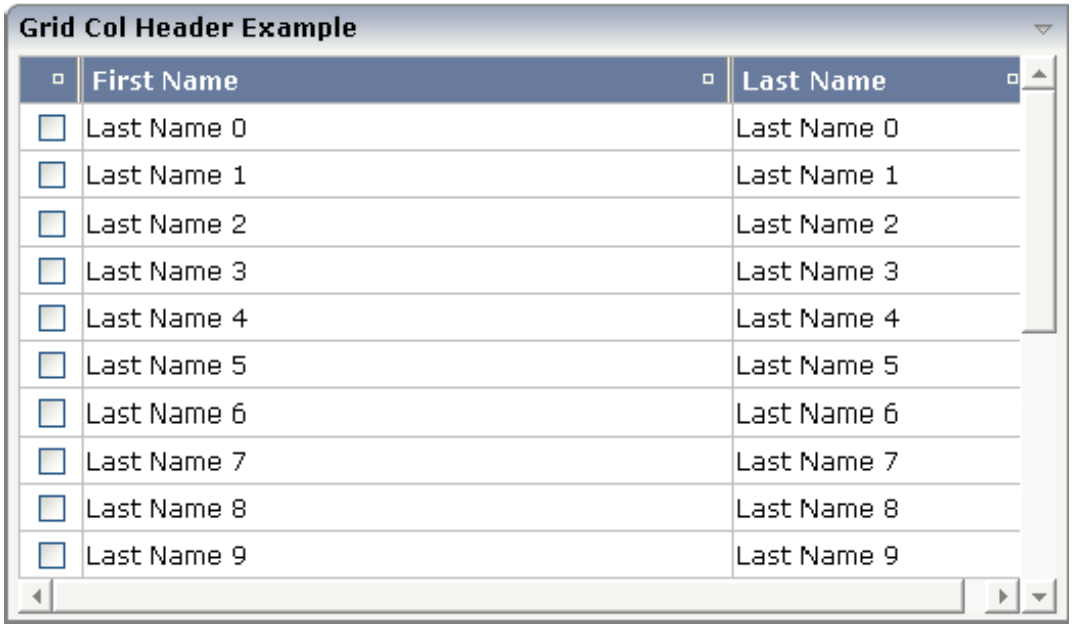
```
                <field valueprop="firstName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <field valueprop="lastName" width="100%" noborder="true"
                       transparentbackground="true">
                </field>
                <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.

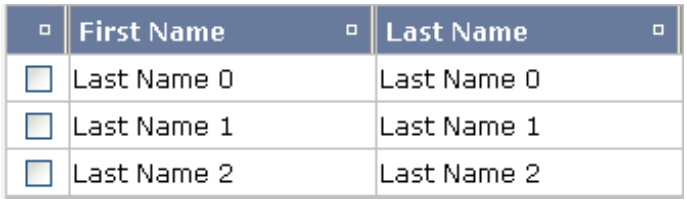
In addition, you can set the widths of the columns by your adapter, e.g. you may store column widths inside your application in order to save the user's column settings and later on reapply the width information.

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.



First Name	Last Name
Last Name 0	Last Name 0
Last Name 1	Last Name 1
Last Name 2	Last Name 2

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:

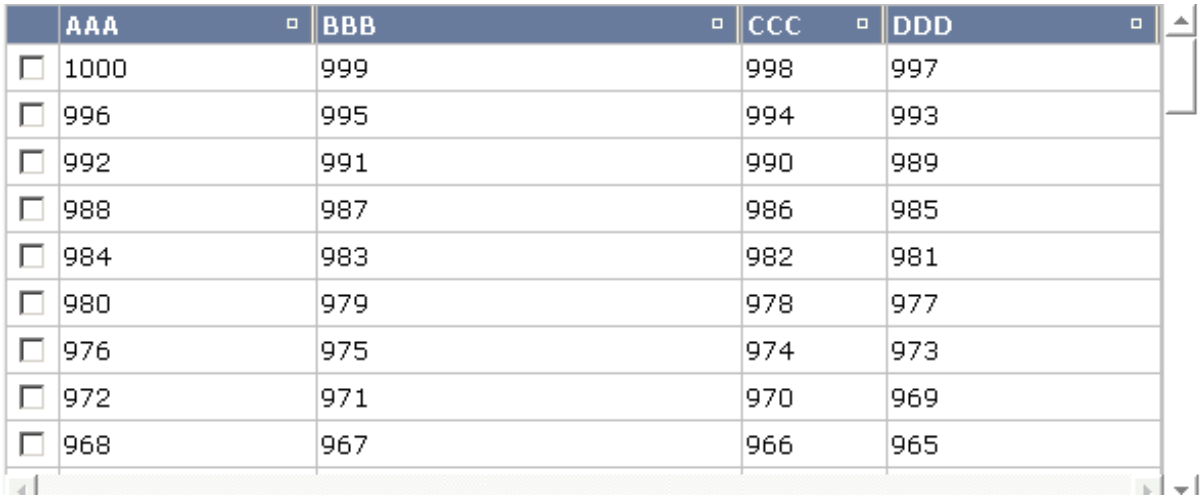
```
<rowtablearea2 griddataprop="lines" rowcount="20" width="100%"
               withborder="true" hscroll="true" firstrowcolwidths="true">
   <tr>
     <label width="25" asheadline="true">
     </label>
     <flexline infoprop="gridheaderline">
     </flexline>
     <hdist>
     </hdist>
   </tr>
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

You see:

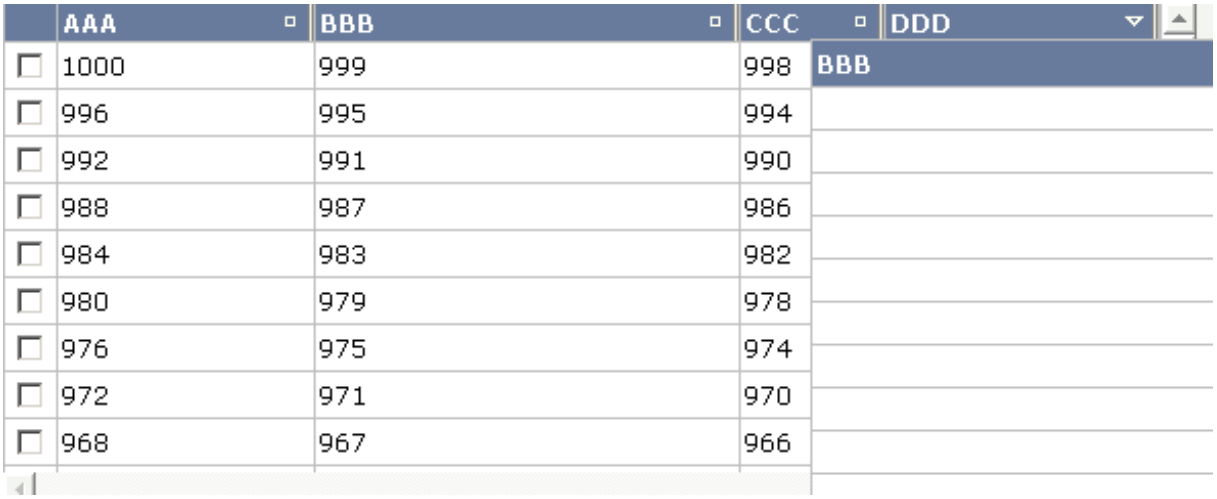
- The ROWTABLEAREA2 definition is set to always follow the column widths of the first row (FIRSTROWCOLWIDTH is set to "true"). The first row of the grid is the row containing the header FLEXLINE control. This means that this row defines the column sizing for the whole grid.
- The header row of the grid 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.

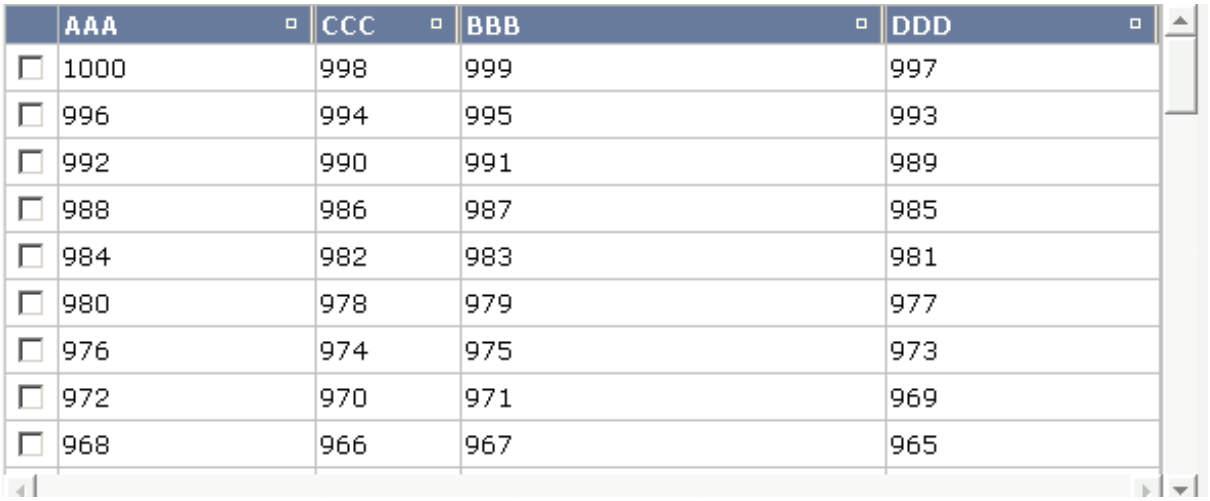
The user can change the sequence of the columns by moving a column header to the position of another header. Example:



	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 percentage 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.	Optional	

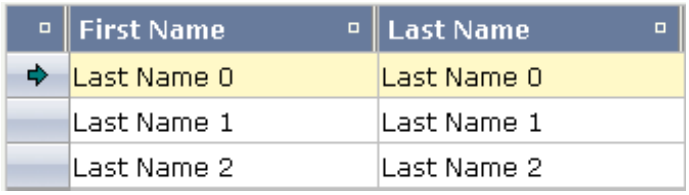
	Either specify the text "hard" by using this TITLE property - or use the TITLETEXTID in order to define a language dependent literal.		
titletextid	Text ID that is passed to the multi language management - representing the tooltip text that is used for the control.	Optional	
withsorticon	Flag that indicates if a small sort indicator is shown within the right corner of the control. Default is TRUE.	Optional	true false
withfrozencolumnsicon	Add an icon to the column header to freeze columns horizontally to the left on click.	Optional	true false
frozentrueicon	Image URL that is shown if the corresponding property value is "true".	Optional	
frozenfalseicon	Image URL that is shown if the corresponding property value is "true".	Optional	
togglestylevariant	Set a value for this attribute if you would like to apply your own sort images. Own sort images must be provided in the subfolder images of your user interface component. For a togglestylevariant ABC the name of the image files must be sort0ABC.gif, sort1ABC.gif, sort2ABC.gif.	Optional	
resizable	Set this to FALSE if the header should not be resizable. 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. specifying "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"	Optional	VAR1 VAR2

	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!		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 language management - representing the tooltip text for the sort indicator.	Optional	
movablecol	Set this attribute to TRUE if you want the columns to be movable at runtime. Default is FALSE. Please notice that only specific controls like FIELD in a grid support movable columns.	Optional	true false
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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 synchronized 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			
nameprop	Name of adapter property that provides as value the text that is shown inside the control.	Optional	
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 multiple rows using a combination of CTRL and click or SHIFT and click.

The SELECTOR control references a boolean 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 percentage 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	

66 Styling Grids

In many applications you would want your grids to have the same style in the whole application. This can be achieved via adapting the corresponding style variables and/or style classes of your style sheet (Adapting the Look & Feel).

The following gives some hints about the most frequent customizations. The NJXDEMOS contain running samples.

General Hints

When to adapt style variables and when to adapt style classes

Many controls have a headline (TEXTGRID*, ROWTABLEAREA2). If you want to change the background color for the headlines in all these controls, simply adapt the variable `@@HEADLINEBACKGROUND@@`.

If you only want to adapt the background color in one specific control, adapt the corresponding style class.

Styling ROWTABLEAREA2, ROWTABLEAREA3 and Headers

How to style the grid headers

As grid headers you usually have LABEL controls with property `asheadline=true`:

```
<tr>
  <label asheadline="true" name="Column 1" ...></label>
  <label asheadline="true" name="Column 2" ...></label>
</tr>
```

Alternatively, you are using GRIDCOLHEADER controls:

```
<tr>
  <gridcolheader name="Column1" ...></gridcolheader>
  <gridcolheader name="Column1" ...></gridcolheader>
</tr>
```

In both cases, the following applies:

- To change the headline background color for multiple controls, simply adapt the variable `@@HEADLINEBACKGROUND@@` in the **General** tab of the Style Sheet Editor tool.
- To change the headline background color only for the headers in ROWTABLEAREA2 and ROWTABLEAREA3 or if you want to do more advanced header styling, adapt the style setting:

```
LABEL
     LABELCellHeadline
```

How to style the sort icons

You can apply your own sort icons by setting the attribute `togglestylevariant` in your GRIDCOLHEADERs:

```
<gridcolheader name="Column1" togglestylevariant="ABC" ...></gridcolheader>
```

As value for `togglestylevariant` use a simple short string. Mostly you would use a subset from your style sheet name.

Copy your own sort icon images to a subfolder named *images* of your user interface component. Adhere to the following naming conventions for the file names:

Sort status	File name	Example
unsorted	sort0<togglestylevariant>.gif	sort0ABC.gif
sort descending	sort1<togglestylevariant>.gif	sort1ABC.gif
sort ascending	sort2<togglestylevariant>.gif	sort2ABC.gif

How to style (background) colors of your grid cells

A simple way to adapt colors for the grid cells is: In the TR or STR of your REPEAT control set the `withalterbackground` property to true:

```
<repeat>
        <str withalterbackground="true" ...>
                <field valueprop="somefield" ...>
```

Now you can simply adapt the variables `@@EVENCELLBACKGROUND@@` and `@@ODDCELLBACKGROUND@@` using the Style Sheet Editor tool. This will also change the background color in TEXTGRID* controls to the same value.

To change the background color only for ROWTABLEAREA2 and ROWTABLEAREA3 or if you want to do more advanced row styling, adapt the style settings:

```
STR
   STRRowEven
   STRRowOdd
```

How to style (background) colors of your header

If your header row consists only of GRIDCOLHEADER controls, you would usually style your header cells as described above. Sometimes you have additional controls like a TOGGLE attribute

in the header row of your grid. In this case, you want to set a specific background color for the whole header row including the cells with the TOGGLE controls.

To style the header row in general, set the property `asheadline` in the TR container of the header as follows:

```
<tr asheadline="true">... </tr>
```

Then you can adapt the style in the style class:

```
TR
   TRHeadline
```

How to set common style attributes like padding for all cells in a grid

If you want to set the same padding value for all cells in the grid adapt the style setting:

```
TABLEAREA
   TABLEAREABorder>tbody>tr>td
```

How to set different style attributes like padding for all header and content cells

If you have resizable headers you usually do not want to have the same style settings of your content cells in the header. For instance, you do not want to set padding attributes in the header but would like to have general padding settings in the content cells.

To style the content classes use the style classes under STR in the Stylesheet Editor tool. For example you could set padding-left and padding-right style attributes for the content cells in the following style classes:

```
STR
   STRRowSelected
   STRRowUnSelected
```

For the header cells you would set different values for the padding-left and padding-right attributes in the following style class:

```
TR
   TRHeadline
```

Since padding values are not inherited down an HTML table in HTML5, you need to additionally set the values for the `padding-left` and `padding-right` attributes to `inherit` in the following style class:

```
TABLEAREA
   TABLEAREABorder>tbody>tr>td
```

That way, you are actually setting different values for the different rows, thus making sure these different values are inherited down the cells underneath the rows.

How to customize the border style of the grid

To adapt general table settings such as the border for the grid, adapt the style classes under TABLEAREA.

67 FLEXGRID - Flexible Grid, Hiding the Grid Complixity for Developers

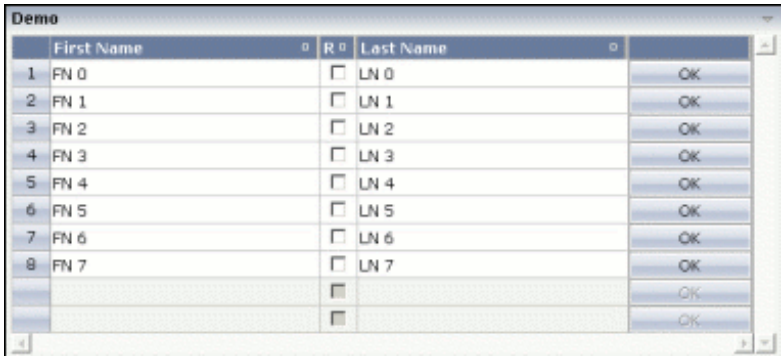
In the previous sections, you saw the basics that make a flexible grid:

- ROWTABLEAREA2, REPEAT, STR as controls for defining a grid structure.
- GRIDCOLHEADER for defining header columns and getting move, resize events.
- FLEXLINE for defining a column's layout (both for header and for items).
- SELECTOR for selecting rows.

Even though each control has its dedicated task and is itself fairly uncomplex, the combination of all controls is not easy for developers to cope with in order to build flexible grids.

The FLEXGRID control is a pre-packaged arrangement of all these controls, combined with a server-side processing that is available using the corresponding `FLEXGRIDInfo` class. With a FLEXGRID control, you can easily (and dynamically) set up the layout of a grid - and all the advantages such as reacting on moving columns are automatically available.

Have a look at the following grid:



It looks like a normal grid - the corresponding layout definition shows the difference:

```
<rowarea name="Demo">
    <flexgrid infoprop="grid" selectprop="selected" rowcount="10" ↩
singleselect="false">
    </flexgrid>
</rowarea>
```

The definition of the grid is very compact - only pointing to a certain property on the server side (`gridinfoprop`), defining a selection property (`selectprop`) and a row count.

The server-side code is also quite simple:

```
package com.softwareag.cis.test40;

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class FlexGrid2Adapter
    extends Adapter
{
    public class MyLine extends SelectableLine
    {
        String m_firstName;
        String m_lastName;
        boolean m_released;
        public String getFirstName() { return m_firstName; }
        public void setFirstName(String firstName) { m_firstName = firstName; }
        public String getLastName() { return m_lastName; }
        public void setLastName(String lastName) { m_lastName = lastName; }
        public boolean getReleased() { return m_released; }
        public void setReleased(boolean released) { m_released = released; }
    }

    FLEXGRIDInfo m_grid = new FLEXGRIDInfo(this);
    public FLEXGRIDInfo getGrid() { return m_grid; }
    public void setGrid(FLEXGRIDInfo value) { m_grid = value; }

    public void init()
    {
        m_grid.clearColumnStructure();
        m_grid.addFieldColumn("firstName","50%","name;First ↩
Name","transparentbackground;true");
        ↩
m_grid.addCheckboxColumn("released","30","name;Rel","transparentbackground;true");
        m_grid.addFieldColumn("lastName","50%","name;Last ↩
Name","transparentbackground;true");
        m_grid.addButtonColumn("100","name;","name;OK");
        for (int i=0; i<8; i++)
        {
            MyLine ml = new MyLine();
            ml.setFirstName("FN " + i);
            ml.setLastName("LN " + i);
            m_grid.getLines().add(ml);
        }
    }
}
```

There is a property `grid` of type `FLEXGRIDInfo` that is referenced by the control. In the `init()` method of the adapter, the grid is prepared: diverse controls are added (the same controls as with FLEXLINE are available for dynamic adding).

Have a look at the following Java statement:

```
m_grid.addFieldColumn("firstName","50%","name;First ↩
Name","transparentbackground;true");
```

There are four parameters that are passed:

- The name of the "valueprop" for the FIELD control that is internally generated.
- The width of the control.
- The additional properties of the GRIDCOLHEADER control that is internally generated as header column.
- The additional properties of the FIELD control that is generated as content.

At any point of time, you can change the column layout inside your adapter by calling the method `clearColumnStructure()` and then recalling the `addField`/`addCheckbox` etc. methods.

FLEXGRID Properties

Basic			
infoprop	Name of the adapter property that provide a FLEXGRIDInfo object that serves the control on server side. The structure of columns is defined within this object using a JAVA API.	Obligatory	
selectprop	Name of the item property that indicates if a grid line is selected.	Obligatory	
rowcount	Number of rows that are rendered inside the control. There are two ways of using this property - dependent on whether you in addition define the HEIGHT property: If you do NOT define the HEIGHT property then the control is rendered with exactly the number of rows that are defined as ROWCOUNT value. If a HEIGHT value is defined in addition (e.g. as percentage value "100%") then the number of rows depend on the actual height of the control. The ROWCOUNT value in this case indicates the maximum number of rows that are picked from the server. You should define this value in a way so that it is not too low - otherwise your grid will not be fully filled. On the other hand it should not be defined too high ("100") because this causes more communication traffic and more rendering effort inside the browser.	Optional	1 2 3 int-value
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a	Optional	100 150 200

	container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		250 300 250 400 50% 100%
vscroll	Definition of the vertical scrollbar's appearance. You can define that scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "scroll".	Optional	auto scroll hidden
withblockscrolling	If switched to "true" then the grid will show small scroll icons by which the user can scroll the grid's content. Scrolling typically is done by using the grid's scrollbar - the scroll icons that are switched on by this property are an additional possibility to scroll.	Optional	true false
showemptylines	Flag that indicates if a line that is not used at the moment is visible. Example: if set to false a grid with rowcount of ten and collection size of seven the last three remaining lines become invisible. Default is true.	Optional	true false
Selector			
selectorwidth	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 percentage 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
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	

Overriding FLEXGRIDInfo

You can override the `FLEXGRIDInfo` class at any time and build up your own, extended class. See the Java API documentation for more details.

68 Sorting Aspects with Grids

Application Designer's grid controls support automated sorting of items.

- TEXTGRID(SSS): by default, you can sort the grid columns by clicking on the corresponding columns header. When using CSVCOLUMN controls, you need to explicitly tell which property is behind which column (property `proprefsprop`).
- ROWTABLEAREA2: by using the GRIDCOLHEADER control, you have the same type of sorting as you have with TEXTGRIDs.

Default Sorting

The default sorting is done in the following way:

- Application Designer sorts the grid collection on the server side.
- Application Designer accesses the grid item objects by reflection or by dynamic access. (See *Binding between Page and Adapter* for more information on dynamic access.)
- Application Designer is taking the data type of the property into consideration when deciding whether to sort lexically or numerically.

Your Own Sorting

You can override the default sorting inside a `TEXTGRIDCollection` or `GRIDCollection`. The following example shows how to do so:

```
package com.softwareag.cis.demoapps;

// This class is a generated one.

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class GridSortAdapter
    extends Adapter
{
    /**
     * Own textgrid collection that overrides the default sort behaviour.
     */
    public class MyTGC extends TEXTGRIDCollection
    {
        /**
         * Own SortInfo-class. In the sort-method you can do "everything you like"
         * for sorting: you can e.g. manipulate the grid collection in any way.
```

```
         */
        public class MySI extends SORTInfo
        {
            public void sort(String sortProperty, boolean ascending)
            {
                outputMessage(MT_SUCCESS,"Sorting: " + sortProperty + ", " + ↩
ascending);
            }
        }

        protected SORTInfo createGridSortInfo()
        {
            MySI result = new MySI();
            return result;
        }
    }

    /**
     * "Normal" lines item object.
     */
    public class LinesItem
    {
        // property >firstName<
        String m_firstName;
        public String getFirstName() { return m_firstName; }
        public void setFirstName(String value) { m_firstName = value; }

        // property >lastName<
        String m_lastName;
        public String getLastName() { return m_lastName; }
        public void setLastName(String value) { m_lastName = value; }
    }

    MyTGC m_lines = new MyTGC();
    public MyTGC getLines() { return m_lines; }

    public void init()
    {
        for (int i=0; i<100; i++)
        {
            LinesItem li = new LinesItem();
            li.setFirstName("FN " + i);
            li.setLastName("LN " + i);
            m_lines.add(li);
        }
    }
}
```

Instead of working on the normal `TEXTGRIDCollection`, you work on a derived one (in the example, this is `MyTGC`). In the derived implementation, you need to overwrite the method

`createGridSortInfo()` returning an object that extends `SORTInfo` (in the example, this is `MySI`). Inside the `SORTInfo` object, you need to implement the `sort(..)` method as shown in the example.

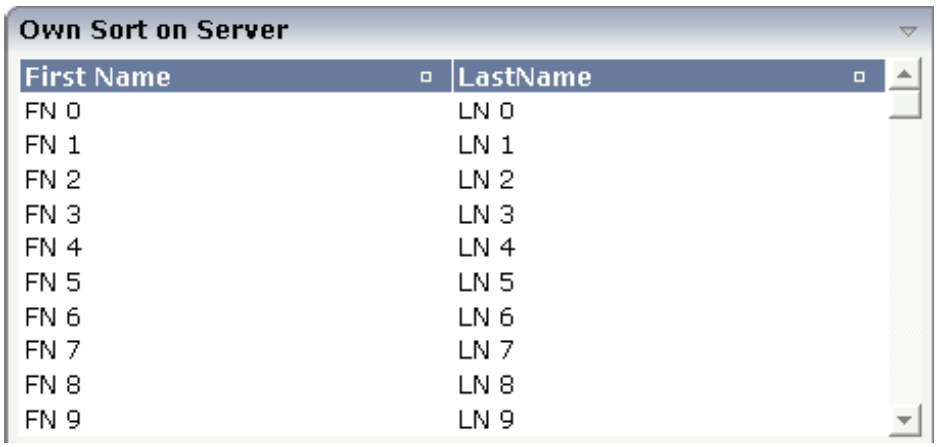
In the example, all classes are defined as inner classes. Of course, the choice whether to use inner classes or normal classes is up to you.

Special Consideration with CSVCOLUMN Controls

The CSVCOLUMN is a dynamic arrangement of columns inside a text grid. The adapter specifies the sequence, width and content of columns at runtime. In order to let Application Designer know how to sort the corresponding items of the `TEXTGRIDCollection`, you need to tell via the CSV-COLUMN property `proprefsprop` which property is behind which column.

When using the default sorting (i.e. no derived `TEXTGRIDCollection` as shown in the previous section), then Application Designer will sort the grid collection by accessing the properties. This means that you have to expose each column property accordingly.

Example:



The layout definition is:

```
<rowarea name="Own Sort on Server">
    <itr takefullwidth="true">
        <textgridsss2 griddataprop="lines" rowcount="10" width="100%">
            <csvcolumn titlesprop="titles" valuesprop="values"
                       widthsprop="widths" proprefsprop="proprefs">
            </csvcolumn>
        </textgridsss2>
    </itr>
</rowarea>
```

The adapter implementation looks like this:

```
package com.softwareag.cis.demoapps;

// This class is a generated one.

import java.util.*;

import com.softwareag.cis.file.CSVManager;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class GridSortAdapter
    extends Adapter
{
    public class LinesItem
    {
        String m_firstName;
        public String getFirstName() { return m_firstName; }
        public void setFirstName(String value) { m_firstName = value; }

        String m_lastName;
        public String getLastName() { return m_lastName; }
        public void setLastName(String value) { m_lastName = value; }

        public String getValues()
        {
            return CSVManager.encodeString(new String[] {m_firstName,m_lastName});
        }

    }

    String m_proprefs = "firstName;lastName";
    public String getProprefs() { return m_proprefs; }

    String m_titles = "First Name;LastName";
    public String getTitles() { return m_titles; }

    String m_widths = "50%;50%";
    public String getWidths() { return m_widths; }

    TEXTGRIDCollection m_lines = new TEXTGRIDCollection();
    public TEXTGRIDCollection getLines() { return m_lines; }

    public void init()
    {
        for (int i=0; i<100; i++)
        {
            LinesItem li = new LinesItem();
            li.setFirstName("FN " + i);
            li.setLastName("LN " + i);
            m_lines.add(li);
        }
```

```
    }
}
```

The `LinesItem` class exposes three properties:

- The `values` property passes back the comma separated string that contains the content of the columns.
- The `firstName` and `lastName` properties are exposed for Application Designer being able to sort the grid collection by property access.

Since Application Designer communicates all simple datatype properties to the client that are part of an accessed object in the scenario above, all three properties are transferred to the UI client - though of course only the `values` property is actually required. The other properties are used for sorting purposes only.

To avoid this, you may use a certain interface `IControlPropertyAccess` with which you can clearly tell Application Designer that certain simple datatype properties - though provided in the implementation - are not relevant to be transferred to the client:

```
/**
 * "Normal" lines item object.
 */
public class LinesItem
    implements IControlPropertyAccess
{
    String m_firstName;
    public String getFirstName() { return m_firstName; }
    public void setFirstName(String value) { m_firstName = value; }

    String m_lastName;
    public String getLastName() { return m_lastName; }
    public void setLastName(String value) { m_lastName = value; }

    public String getValues()
    {
        return CSVManager.encodeString(new String[] {m_firstName,m_lastName});
    }

    public String[] findPropertiesNotToBeCollected()
    {
        return CSVManager.decodeString(m_proprefs);
    }
}

String m_proprefs = "firstName;lastName";
public String getProprefs() { return m_proprefs; }
```

If you do not want to provide for the fine granular properties at all (`firstName`, `lastName`), then you still can use the possibility of doing the sorting completely on your own as shown in the pre-

vious section. In this case, the property references passed by PROPREFSPROP are just strings thay are communicated to the `sort(..)` method when the user sorts a grid.

69 Background Information on Grids

If you are interested in more background information on how grids internally work, see *Binding between Page and Adapter* in the *Java Pages Development*. Basically, a grid maps to an array of data internally - and the array of data is provided by the corresponding grid collections (`GRIDCollection`, `TEXTGridCollection`).

V Working with Trees

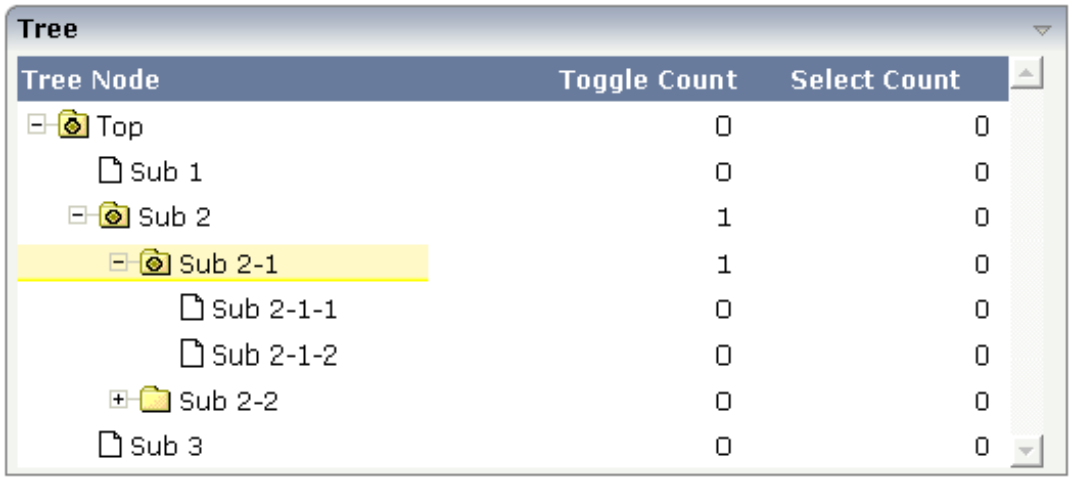
This part shows you how to work with trees and tree nodes. The information is organized under the following headings:

70 Basics

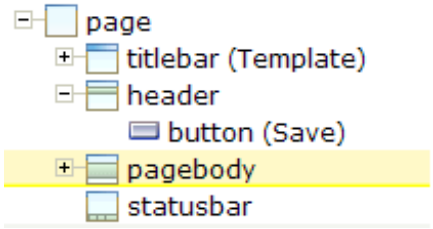
Types of Trees

The following controls are available for building trees:

- **TREENODE3**
  This control displays a single tree node. It can be put into the normal control grid (ROWTABLEAREA2), and can consequently be combined with any other control (for example, FIELD, TEXTOUT, etc.).



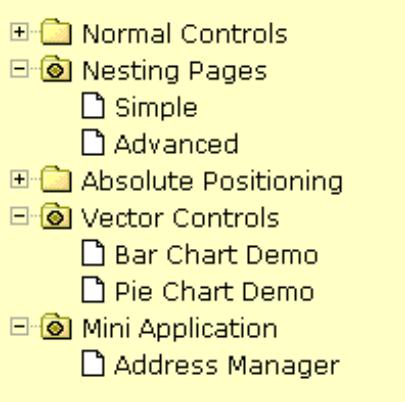
  Of course, you do not have to combine it with other controls. You can also use it "stand-alone" inside a ROWTABLEAREA2 grid:



  As with the normal ROWTABLEAREA2 management, only these items are transferred from the server to the client which are currently visible. Items which are collapsed or which are not in the visible area of the client, are not transferred.

  All scrolling of items and all toggling of items (opening/collapsing) goes through the server.

- **CLIENTTREE**
  This control represents a whole tree. You cannot add further controls into the tree node lines.



The data which is displayed inside the tree is transferred from the server to the client in one step - always the whole tree. The data is transferred when opening a page or when the tree data in the server is updated.

All scrolling of items and all toggling of items (opening/collapsing) is done in the client without going back to the server.

When to Use Which Type

Use the TREENODE3 control inside the control grid ROWTABLEAREA2 in the following cases:

- High number of tree nodes.
- Tree nodes are not loaded from the beginning, but step by step.
- Data in the tree is exchanged/updated quite often.

Use the CLIENTTREE control in the following cases:

- Low number of tree nodes (100).
- High interactivity requirements for toggling nodes.
- Data in the tree is rather static. It is loaded once into the client, and afterwards it is not changed anymore.

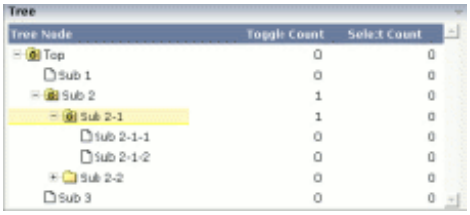
Example: in the Application Designer environment, the tree controls are used in the following way:

- In the workplace, a CLIENTTREE is loaded: the number of nodes is quite low, the tree represents a menu which is rather static.
- In the Layout Painter, a TREENODE2 in a ROWTABLEAREA2 is used for representing the XML control tree: the number of items can be quite high, the update rate of the tree data is very high.

71 TREENODE3 in Control Grid (ROWTABLEAREA2)

Example

The following image shows an example for a tree management:



The grid contains three columns: the first column shows the tree node, the other two columns display some text information.

The XML layout definition is:

```
<rowarea name="Tree">
    <rowtablearea2 griddataprop="treeGridInfo" rowcount="8" width="500" ↩
withborder="false">
        <tr>
            <label name="Tree Node" width="200" asheadline="true">
            </label>
            <label name="Toggle Count" width="100" asheadline="true"
                   labelstyle="text-align:right">
            </label>
            <label name="Select Count" width="100" asheadline="true"
                   labelstyle="text-align:right">
            </label>
        </tr>
        <repeat>
            <tr>
                <treenode3 width="200" withplusminus="true"
                           imageopened="images/fileopened.gif"
                           imageclosed="images/fileclosed.gif"
                           imageendnode="images/fileendnode.gif">
                </treenode3>
                <textout valueprop="toggleCount" width="100" align="right">
                </textout>
                <textout valueprop="selectCount" width="100" align="right">
                </textout>
            </tr>
        </repeat>
    </rowtablearea2>
</rowarea>
```

You see that the TREENODE3 control is placed inside the control grid just as a normal control. There are certain properties available which influence the rendering: in the example, the name of

the tree node images is statically overwritten. The flag `withplusminus` is set to true - consequently, small "+"/"-" icons are placed in front of the node.

The corresponding adapter code is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.NODEInfo;
import com.softwareag.cis.server.util.TREECollection;

public class tree_01Adapter
    extends Adapter
{
    // class >TreeGridInfoItem<
    public class Item extends NODEInfo
    {
        int m_toggleCount = 0;
        int m_selectCount = 0;

        public Item(String text)
        {
            super(text);
        }
        public void reactOnToggle() { m_toggleCount++; }
        public void reactOnSelect() { m_selectCount++; }
        public int getToggleCount() { return m_toggleCount; }
        public int getSelectCount() { return m_selectCount; }
    }

    // property >treeGridInfo<
    TREECollection m_treeGridInfo = new TREECollection();
    public TREECollection getTreeGridInfo() { return m_treeGridInfo; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_treeGridInfo = new TREECollection();
        Item item = new Item("Top");
        m_treeGridInfo.addTopNode(item,false);
        m_treeGridInfo.addSubNode(new Item("Sub 1"),item,true,false);
        Item subItem = new Item("Sub 2");
        m_treeGridInfo.addSubNode(subItem,item,false,false);
        Item subItem21 = new Item("Sub 2-1");
        m_treeGridInfo.addSubNode(subItem21,subItem,false,false);
        m_treeGridInfo.addSubNode(new Item("Sub 2-1-1"),subItem21,true,false);
        m_treeGridInfo.addSubNode(new Item("Sub 2-1-2"),subItem21,true,false);
        Item subItem22 = new Item("Sub 2-2");
        m_treeGridInfo.addSubNode(subItem22,subItem,false,false);
        m_treeGridInfo.addSubNode(new Item("Sub 2-2-1"),subItem22,true,false);
        m_treeGridInfo.addSubNode(new Item("Sub 3"),item,true,false);
```

```
        // open top node
        m_treeGridInfo.toggleNode(item);
    }
}
```

The grid collection is an instance of the class `TREECollection` from the package `com.softwareag.cis.server.util`. (Remember that the class `GRIDCollection` is used for normal grids.) The `TREECollection` has all functions that are required for:

- server-side scrolling,
- selecting tree nodes,
- opening and closing tree nodes.

The items of the tree collection are derived from a predefined class `NODEInfo` from the package `com.softwareag.cis.server.util`. By overwriting the methods `reactOnToggle()` and `reactOnSelect()`, you can react on user interaction. Each tree node is represented by one single instance of this item class.

The tree is built inside the `init()` method of the adapter. For filling the tree, the following methods of the `TREECollection` class are used:

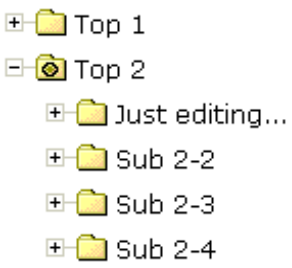
- `addTopNode()`
- `addSubNode()`

See the JavaDoc documentation for more information on these methods and other methods for manipulating the tree.

The tree can be filled “all at once” - as shown in this example - or loaded step by step on the server side. For example, the tree can be extended in the `reactOnToggle()` method when a node is being opened.

Editing the Text of the Tree Node

You may already have seen the property `withtextinput` which you may set to "true". If doing so, then the user can double-click on a tree node and edit the node's text:



By pressing TAB or ENTER, the input is taken over into the tree node and is by default transferred to the adapter with the next request (e.g. when a button is chosen on the screen). Users can also use the ESC key - in this case, the tree node is set back to its former value. The server-side adapter can pick the text by the normal `getText()` method which is available in the tree node and is implemented on `NODEInfo` level.

If you want your server program to be explicitly notified by the text change, then override the `setText()` method inside your tree node implementation:

```
...
...
...
public class TreeItem extends NODEInfo
{
    ...
    ...
    ...
    public void setText(String value)
    {
        super.setText(value);
        // do your own implementation here
        outputMessage(MT_SUCCESS,"Node text changed: " + value);
    }

    ...
    ...
}
...
...
...
```

You can also explicitly define the point of time when the text change in the user interface is transferred to the server. As with normal input controls (FIELD, CHECKBOX, etc.), the text change is by default registered in the browser, but does not trigger an immediate transfer to the server-side adapter. By using the property `flush`, you can control this: setting this property to "server" will immediately synchronize the client with the adapter; setting it to "screen" will immediately synchronize inside the browser.

There is still one issue: inside the tree node item, there is the method `setDisableTextInput(boolean)`. Calling this method will switch off the editing behavior for this tree node. Consequently, you can explicitly define tree nodes that allow to edit text and others that do not allow to do so. In case the user double-clicks onto a node that is explicitly set to be not editable, the text will be displayed in disabled format so that the user receives visual feedback that this operation is not supported for this node.

Embedding Controls into TREENODE3

It is possible to add further controls into the tree node. The typical cases are:

- a check box,
- an icon,
- a toggle control.

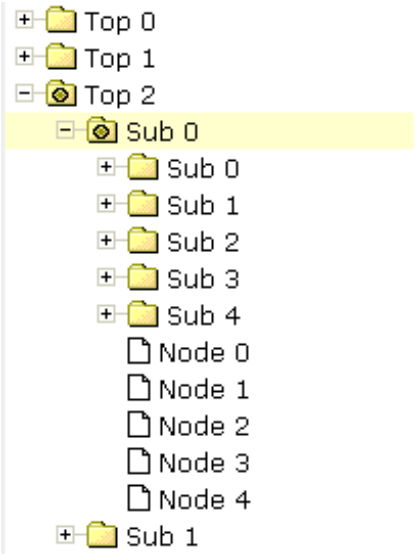
The toggle control offers the possibility to manipulate a boolean value - it is similar to a check box, but allows to explicitly define a "true-image" and a "false-image". When using the toggle control inside the tree node, there is one useful feature: the toggle control allows to be defined in such a way that it show three different images:

- "true-image"
- "false-image"
- "partial-image"

In trees, you typically have selections in which you want to select by toggle control one item and all of its subitems. The same goes for deselecting. But you also want to be able to express that inside one node, there are some selected subitems, but not all subitems are selected. The toggle control exactly matches these requirements. For more information, see the description of the **TOGGLE** control.

Loading Large Trees - Step by Step

In the example at the beginning of this TREENODE3 section, the whole `TREECollection` inside the adapter was filled in one step. The following example shows how to dynamically load elements into a tree that gets larger and larger due to the user's navigation in the tree.



Every time the user opens a folder, the folders "Sub 0" to "Sub 4" and the end nodes "Node 0" to "Node 4" are input into the tree hierarchy:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.NODEInfo;
import com.softwareag.cis.server.util.TREECollection;

public class tree_03Adapter
    extends Adapter
{
    // class >TreeItem<
    public class TreeItem extends NODEInfo
    {
        boolean m_subNodesAvailable = false;
        public TreeItem(String text)
        {
            super(text);
        }
        public void reactOnSelect() {}
        public void reactOnToggle()
        {
            if (m_subNodesAvailable == false)
            {
                m_subNodesAvailable = true;
                addSubNodesFor(this);
            }
        }
    // property >tree<
    TREECollection m_tree = new TREECollection();
    public TREECollection getTree() { return m_tree; }
```

```
    public void init()
    {
        TreeItem top;
        for (int i=0; i<10; i++)
        {
            top = new TreeItem("Top " + i);
            m_tree.addTopNode(top,false);
        }
    }

    private void addSubNodesFor(TreeItem top)
    {
        TreeItem sub;
        for (int i=0; i<5; i++)
        {
            sub = new TreeItem("Sub " + i);
            sub.setDisableTextInput(true);
            m_tree.addSubNode(sub,top,false,false);
            sub.setOpened(TREECollection.ST_CLOSED);
        }
        for (int i=0; i<5; i++)
        {
            sub = new TreeItem("Node " + i);
            m_tree.addSubNode(sub,top,true,false);
        }
    }
}
```

Inside the tree node class (`TreeItem`), a boolean member `m_subNodesAvailable` indicates whether subnodes for this instance have already been loaded. In the method `reactOnToggle()`, new nodes are added by calling the `addSubNodesFor(...)` method - depending on the `m_subNodesAvailable` value.

If you are “very eager” or if you have for some good reason to be very strict in memory-house-keeping, then you could also remove all subnodes of a node when the node is closed.

Drag-and-Drop Inside a TREENODE3 Tree

Implementing drag-and-drop inside your tree is easy - you just have to do two things:

- Set the `enabledrag` property to "true" inside the TREENODE3 definition.
- Add a method `reactOnContextMenuRequestDragTarget()` into your node class and implement your reaction.

The tree node inside your page will automatically offer the following behavior: when selecting one or more nodes, you can click on the node's text, drag the nodes, and drop them onto another node's text.

Sorry for the name `reactOnContextMenuRequestDragTarget()` - it assumes that you ought to open a context menu, but you can do any other reaction as well. Of course, it is a nice feature to offer a context menu when the user drops items onto another item - showing the user what functions can be executed with the dropped items.

The following simple demo shows an example in which the node on which other nodes are dropped outputs the text of the dropped nodes:

```
...
...
...
// class >TreeItem< representing node object
public class TreeItem extends NODEInfo
{
    ...
    ...
    ...
    public void reactOnContextMenuRequestDragTarget()
    {
        // iterate through selected tree nodes and concatenate text
        NODEInfo[] selItems = m_tree.findSelectedItems();
        StringBuffer sb = new StringBuffer();
        for (int i=0; i<selItems.length; i++)
        {
            if (i != 0) sb.append(", ");
            sb.append(selItems[i].getText());
        }
        outputMessage(MT_SUCCESS,"Drop result: " + sb.toString());
    }
    ...
    ...
    ...
}
...
...
...
```

In the method `reactOnContextMenuRequestDragTarget`, the selected items are identified by using the tree collection's method `findSelectedItems()`.

Note: With the TREENODE3 property `singleselect`, you can change the tree from its default “single select mode” into “multi select mode”.

Dynamic Setting of Tree Icons

There are three ways to define icons for a tree node:

- No definition at all. The nodes will be rendered with the default icons.
- Fixed definition of icons. Using the TREENODE3 properties `imageopened`, `imageclosed` and `imageendnode`, you can define the icons to be used.
- Dynamic definition of icons. Each node can be assigned an own icon. The icon is defined by a property of the server-side node object. The name of the property is defined inside the TREENODE3 property `imageprop`.

Have a look at the following example:

Each of the subnodes has its own icon, depending on its category. (An individual icon can also be used for folder nodes.)

The XML layout definition is:

```
<page model="com.softwareag.cis.demoapps.TreeDynamicIconsAdapter">
    <titlebar name="Tree with dynamic Icons">
    </titlebar>
    <pagebody>
        <rowarea name="Tree Demo">
            <rowtablearea2 griddataprop="lines" rowcount="10" width="100%" ↩
withborder="false">
                <repeat>
                    <tr>
                        <treenode3 width="100%" imageprop="imageName">
                        </treenode3>
                    </tr>
```

```
                </repeat>
            </rowtablearea2>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

In the layout definition, you see that the TREENODE3 property `imageprop` points to the property `imageName`.

The adapter code is:

```
package com.softwareag.cis.demoapps;

// This class is a generated one.

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class TreeDynamicIconsAdapter
    extends Adapter
{
    public class LinesItem extends NODEInfo
    {
        public LinesItem(String text, String imageName)
        {
            super(text);
            m_imageName = imageName;
        }

        String m_imageName;
        public String getImageName() { return m_imageName; }
        public void setImageName(String value) { m_imageName = value; }

        public void reactOnSelect()
        {
        }

        public void reactOnToggle()
        {
        }
    }

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

    public void init()
```

```
    {
        LinesItem top = new LinesItem("Printers",null);
        m_lines.addTopNode(top,false);
        m_lines.addSubNode(new LinesItem("LPT1 - ↩
Printer","images/print.gif"),top,true,true);
        m_lines.addSubNode(new LinesItem("COM - ↩
Printer","images/print.gif"),top,true,true);
        top = new LinesItem("Disks",null);
        m_lines.addTopNode(top,false);
       m_lines.addSubNode(new LinesItem("A: drive","images/save.gif"),top,true,true);
       m_lines.addSubNode(new LinesItem("B: drive","images/save.gif"),top,true,true);
        top = new LinesItem("Pencils",null);
        m_lines.addTopNode(top,false);
        m_lines.addSubNode(new LinesItem("My ↩
pencil","images/editdisabled.gif"),top,true,true);
    }

}
```

The `imageName` property is implemented on tree node level (i.e. as property inside the inner class `LinesItem`). In the `init` method, the property is defined to be "null" for the top nodes and to hold a value for the leaf nodes.

Again: you could also define an individual icon for the top nodes - in the same way you do it for the leaf nodes. The example above shows that the different ways of assigning icons build on one another: if the dynamic icon is not passed (as done with the top nodes), then these icons are selected that are defined with `imageopened`/`imageclosed`/`imageendnode` properties. If these are not defined (as in the example), the default icon is used.

Properties

Basic			
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 percentage 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	Optional	1 2 3 int-value

	define a width of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
withplusminus	If set to "true" then +/- Icons will be rendered in front of the tree items.	Optional	true false
withlines	If set to "true" then the tree elements are connected with one another by gray lines. Please pay attention: PAGE layouts (Java), if switching this property to "true" then you have to create the instance of your server side TREECollection object with a special constructor: Example: TREECollection m_tree = new TREECollection(true)	Optional	true false
withtooltip	If set to "true" then the text of an item is also available as tool tip. Use this option in case you expect that the horizontal space of the item will not be sufficient to display the whole text of the item.	Optional	true false
withtextinput	If set to "true" then the tree node can also be edited. Editing is started when the user double clicks the node. The text that is input is passed into the property "text" which is implemented in the default NODEInfo implementation.	Optional	true false
imageopened	Image of a tree node that has subnodes and that is currently showing its nodes. The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
imageclosed	Image of a tree node that has subnodes and that is currently not showing its nodes. The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
imageendnode	Image of a tree node that is an end node (leaf node). The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
singleselect	If set to "true" then only one item can be selected. If set to "false" then multiple icons can be selected.	Optional	true false
directselectevent	Event that represents a tree node selection. A tree node selection is done when the user clicks/doubleclicks on the tree node text.	Optional	ondblclick

	In this case the select() method is called in the corresponding node object on server side.		onclick
directselectelement	If set to "textonly" only user clicks on the tree node text will select the node. If set to "allspace" also user clicks outside the area occupied by the node text will select the node.	Optional	textonly allspace
selectionstylevariant	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2
textstylevariant	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2
pixelshift	Number of pixels that each hierarchy level is indented. If not defined then a standard is used.	Optional	1 2 3 int-value
pixelshiftendnode	Number of pixels that end nodes are indented. If not defined then a standard is used.	Optional	1 2 3 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.	Optional	1 2 3

	The property only makes sense in table rows that are synchronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.		4 5 50 int-value
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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
pixelheight	Height of the control in pixels.	Optional	1 2 3 int-value
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
Binding			
imageprop	Name of property of the item objects that provides for a image for the tree node. Each node may provide for its own image, e.g. dependent on the type of node. If the adapter property passes back "null" then the image is taken from the static definitions that you may parallely do by	Optional	

	using the properties IMAGEOPENED, IMAGECLOSED and IMAGEENDNODE.		
focusedprop	Name of property of the item objects - representing the individual rows of the collection - that indicates if the row receives the keyboard focus. Must be of type "boolean"/ "Boolean". If more than one lines are returning "true" the first of them is receiving the focus.	Optional	
flush	Flush behaviour when using the possibility of having editable tree nodes. If double clicking on the tree node then you can edit its content. The FLUSH property defines how the browser behaves when leaving the tree node's input field: If not defined ("") then nothing happens - the changed tree node text is communicated to the server side adapter object with the next roundtrip. If defined as "server" then immediately when leaving the field a roundtrip to the server is initiated - in case you want your adapter logic to directly react on the item change. If defined as "screen" then the changed tree node text is populated inside the page inside the front end.	Optional	screen server
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
tooltipprop	Name of property of the item objects that provides for a text that is shown if the user moves the mouse over the tree item (tooltip).	Optional	
validdraginfosprop	Name of a property that contains a 'comma separated list' of valid drag informations.	Optional	
Drag and Drop			
enabledrag	If set to true then drag and drop is enabled within the tree.	Optional	true false

72 CLIENTTREE

Example

The following example shows a simple client tree:

The XML layout definition is:

```
<rowarea name="Clienttree">
    <clienttree treecollectionprop="tree" height="200" withplusminus="true"
                treestyle="background-color:#FEFEEE">
    </clienttree>
</rowarea>
```

In this example, the client tree is directly put as row into the ROWAREA container. The property `treecollectionprop` contains a reference to the property `tree` which contains the net data of the tree. With the property `treestyle`, an explicit background color is set.

The Java code of the adapter is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.NODEInfo;
import com.softwareag.cis.server.util.TREECollection;

public class ClientTreeAdapter
    extends Adapter
{
    // class >TreeItem<
    public class TreeItem extends NODEInfo
    {
        public TreeItem(String text)
        {
```

```
            super(text);
        }

        public void reactOnSelect()
        {
            outputMessage("S", "Node " + getText() + " was selected.");
        }

        public void reactOnToggle()
        {
            // TODO Auto-generated method stub
        }
    }

    // property >tree<
    TREECollection m_tree = new TREECollection();
    public TREECollection getTree() { return m_tree; }
    public void setTree(TREECollection value) { m_tree = value; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        TreeItem file = new TreeItem("File");
        m_tree.addTopNode(file, false);
        m_tree.toggleNode(file); // open this file node to open immediately
        TreeItem news = new TreeItem("New");
        m_tree.addSubNode(news, file, false, false);
        TreeItem project = new TreeItem("Project");
        m_tree.addSubNode(project, news, true, false);
        TreeItem packages = new TreeItem("Package");
        m_tree.addSubNode(packages, news, true, false);
        TreeItem classes = new TreeItem("Class");
        m_tree.addSubNode(classes, news, true, false);
        TreeItem close = new TreeItem("Close");
        m_tree.addSubNode(close, file, true, false);
        TreeItem closeAll = new TreeItem("Close All");
        m_tree.addSubNode(closeAll, file, true, false);
        TreeItem save = new TreeItem("Save");
        m_tree.addSubNode(save, file, true, false);
        TreeItem saveAll = new TreeItem("Save All");
        m_tree.addSubNode(saveAll, file, true, false);
        TreeItem print = new TreeItem("Exit");
        m_tree.addSubNode(print, file, true, false);
        TreeItem exit = new TreeItem("Exit");
        m_tree.addSubNode(exit, file, true, false);
        TreeItem edit = new TreeItem("Edit");
        m_tree.addTopNode(edit, false); // open this file node to open immediately
        m_tree.toggleNode(edit);
        TreeItem undo = new TreeItem("Undo");
        m_tree.addSubNode(undo, edit, true, false);
    }
}
```

You see that the implementation of the server-side representation of the client tree control is using the same mechanisms as the one which you got to know when the TREENODE2 control was explained.

The difference is:

- The `reactOnToggle()` method needs not to be implemented. There is no explicit toggle event passed to the server. The toggling is done completely on the client side.

You also see that the top nodes are toggled immediately via the `TREECollection` API after creation:

```
MyNODEInfo edit = new MyNODEInfo("Edit");
m_tree.addTopNode(edit, false); // open this file node to open immediately
m_tree.toggleNode(edit);
```

Reason: by default, foldable tree nodes are opened with "closed" status. If they are toggled in the creation process, then they are already open when the tree is shown the first time.

Properties

Basic			
treecollectionprop	Name of adapter property representing the tree of items on server side. The property must be of type "TREECollection". Please view in Java API Documentation for more information.	Optional	
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%

comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
withplusminus	If set to "true" then +/- Icons will be rendered in front of the tree items.	Optional	true false
withtooltip	If set to "true" then the text of an item is also available as tool tip. Use this option in case you expect that the horizontal space of the item will not be sufficient to display the whole text of the item.	Optional	true false
selectionvisible	If set to "true" then the clicked item will also marked with a certain background color. The background color is defined by the style sheet settings.	Optional	true false
singleselect	If set to "true" then only one item can be selected. If set to "false" then multiple icons can be selected.	Optional	true false
imageopened	Image of a tree node that has subnodes and that is currently showing its nodes. The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
imageclosed	Image of a tree node that has subnodes and that is currently not showing its nodes. The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
imageendnode	Image of a tree node that is an end node (leaf node). The image either is defined statically by this property or also may be defined dynamically - see the corresponding properties defined with this control.	Optional	
treestyle	Style (following cascading style sheet definitions) that is directly passed to the background area of the client tree. You can manipulate e.g. the colour of the tree's background. The style can also be set dynamically by specifying the property TREESTYLEPROP.	Optional	
selectionstylevariant	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 definition and use them multiple times by addressing them via the "stylevariant" property. CIS currently offers two variants "VAR1" and "VAR2" but does not predefine any semantics behind - this is up to you!	Optional	VAR1 VAR2

hscroll	Definition of the horizontal scrollbar's appearance. You can define that the scrollbars only are shown if the content is exceeding the control's area ("auto"). Or scrollbars can be shown always ("scroll"). Or scrollbars are never shown - and the content is cut ("hidden"). Default is "auto".	Optional	auto scroll hidden
pixelshift	Number of pixels that each hierarchy level is indented. If not defined then a standard is used.	Optional	1 2 3 int-value
pixelshiftendnode	Number of pixels that end nodes are indented. If not defined then a standard is used.	Optional	1 2 3 int-value
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
withleftpadding	Flag that indicates if the control has a 10 pixel padding on left side. Default is true.	Optional	true false
Binding			
treecollectionprop	(already explained above)		
dynamicloading	If set to "true" then you indicate to the tree control that not all tree information may be loaded when initializing the tree (i.e. the tree collection on server side). As consequence the tree control will pass the "toggle-event" to the server - in case the subnodes of a certain nodes are not yet loaded.	Optional	true false

	In the case the toggle event is passed to the server, the method onToggle() is called inside the tree item.		
imageopenedprop	Property of the tree item object that provides for the image URL which is shown for opened tree nodes or end tree nodes. The value may be different from tree node to tree node -each tree node may have an own image.	Optional	
imageclosedprop	Property of the tree item object that provides for the image URL which is shown for closed tree nodes. The value may be different from tree node to tree node -each tree node may have an own image.	Optional	
treestyleprop	Property of the adapter object that dynamically provides for a style value that is passed to the control's area (background of the client tree). You can as consequence e.g. define the background-colour of the tree dependent on your server side logic.	Optional	
treeclassprop	Name of adapter property that passes back the name of a style sheet class that is taken to render the client tree's background area. - Similar to the property TREESTYLEPROP, but now a style class is passed, not the style itself.	Optional	
tooltipprop	Name of property of the item objects that provides for a text that is shown if the user moves the mouse over the tree item (tooltip).	Optional	
oncontextmenumethod	Name of the method on adapter level that is called if the user presses the right mouse button in an empty area of the client tree. Note: if the user presses right mouse button on a node then the method "reactOnContextMenuRequest()" is called in the item object.	Optional	
ondblclickmethod	Name of the method that is called when the user double clicks.	Optional	
directselectevent	Event that represents a tree node selection. A tree node selection is done when the user clicks/doubleclicks on the tree node text. In this case the select() method is called in the corresponding node object on server side.	Optional	ondblclick onclick
focusedprop	Name of property of the item objects - representing the individual rows of the collection - that indicates if the row receives the keyboard focus. Must be of type "boolean"/ "Boolean". If more than one lines are returning "true" the first of them is receiving the focus.	Optional	
Drag and Drop			
enabledrag	If set to true then drag and drop is enabled within the tree.	Optional	true

			false

VI Working with Menus

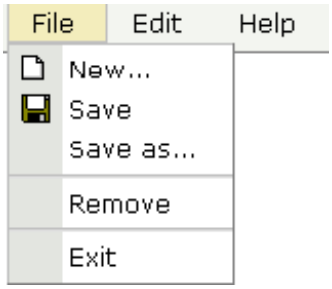
Menus are used to arrange a number of functions in a structured way.

The information provided in this part is organized under the following headings:

73 Types of Menus

The following menu controls are available:

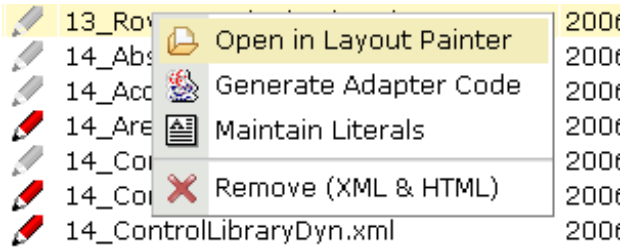
- **MENU**
  This is the typical drop-down menu:



- **DLMENU**
  This is a double-line menu representing a two-level hierarchy. It can be found quite often in web applications.

  When clicking an item in the first line, the corresponding subitems are shown in the second line.

- **Context Menu**
  This is a menu which appears in certain controls (tree controls, grid controls) when the user presses the right mouse button.



All menu controls are completely configured by a corresponding adapter implementation. This means:

- The structure of the menu and its menu nodes is not statically defined but is dynamically derived by some adapter program logic. For example, you can build a personalized menu taking the user's rights into consideration.
- Menu information can be dynamically updated during runtime.

74 MENU

Example

The example looks as follows:

When clicking on a menu item for which a function has been defined, then the name of the function is displayed in the status bar.

The XML layout definition is:

```
<page model="Menue_01_Adapter">
    <titlebar name="Menu Demo">
    </titlebar>
    <header align="left" withdistance="false">
        <menu menucollectionprop="menuData" width="100">
        </menu>
    </header>
    <pagebody>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

In this example, the menu is embedded in the header. By the property `menucollectionprop`, it is bound to the adapter property `menuData`.

The Java code of the adapter is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.MENUNODEInfo;
import com.softwareag.cis.server.util.TREECollection;

public class Menue_01_Adapter
    extends Adapter
{
    // class >MenuDataItem<
    public class MenuDataItem extends MENUNODEInfo
    {
        public MenuDataItem(String text)
        {
            super(text);
        }
        public MenuDataItem(String text, String image)
        {
            super(text, image);
        }
        public void reactOnSelect()
        {
            outputMessage("S",getText() + " was called");
        }
    }

    // property >menuData<
    TREECollection m_menuData = new TREECollection();
    public TREECollection getMenuData() { return m_menuData; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
```

```
        MenuDataItem top;
        top = new MenuDataItem("File");
        m_menuData.addTopNode(top,false);
        m_menuData.addSubNode(new MenuDataItem("New...","images/new.gif"),
        top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Save","images/save.gif"),
        top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Save as..."),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("&SEPARATOR"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Remove"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("&SEPARATOR"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Exit"),top,true,false);
        top = new MenuDataItem("Edit");
        m_menuData.addTopNode(top,false);
        m_menuData.addSubNode(new MenuDataItem("Undo"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("&SEPARATOR"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Cut"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Copy"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("Paste"),top,true,false);
        top = new MenuDataItem("Help");
        m_menuData.addTopNode(top,false);
        m_menuData.addSubNode(new MenuDataItem("Online Help"),top,true,false);
        m_menuData.addSubNode(new MenuDataItem("About"),top,true,false);

    }
}
```

The member `m_menuData` holds an object of the instance `TREECollection`, which is defined in package `com.softwareag.cis.server.util`. The tree collection holds the menu items.

Each item is represented by an instance of the class `MyMENUNODEInfo`. The class itself is derived from the class `NODEInfo` in the package `com.softwareag.cis.server.util`. In the own class definition, the `reactOnSelect()` method is overwritten.

In the `init()` method of the class, the tree collection is assembled. Note that it can be reassembled at any point of time.

Separators

As you see in the example, separators can be added into the menu just as normal tree nodes, holding a special text "&SEPARATOR".

Properties

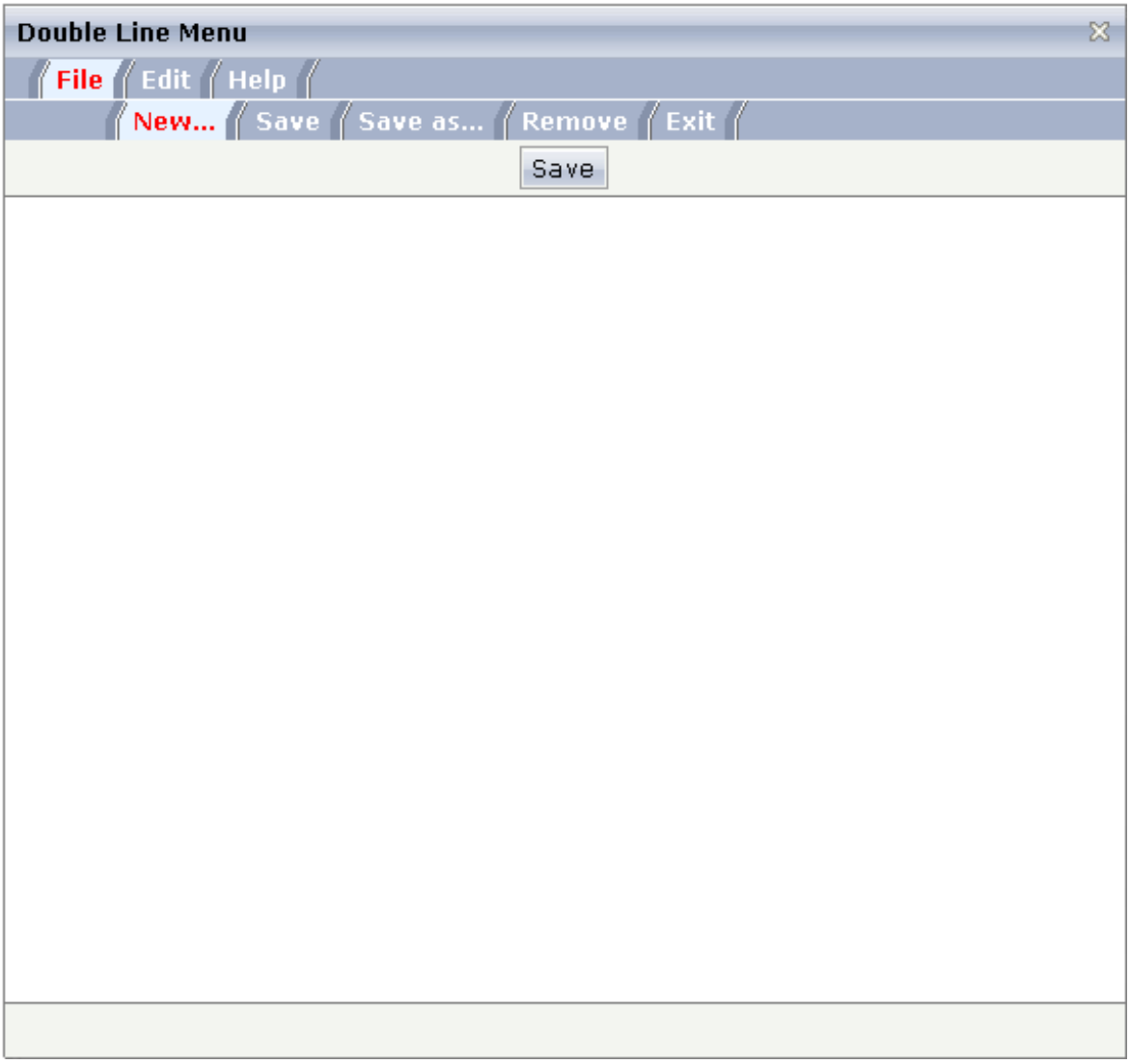
Basic			
menucollectionprop	Name of adapter property that represents the menu's item hierarchy on server side. The property must be of type "TREECollection". Each menu item is represented by a tree node (subclassed from "NODEInfo") within the collection.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
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 percentage 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%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	

toggleimage	URL of the image that is shown on the right end of a menu item, if this item contains subitems. If not explicitly defined then a default icon is used.	Optional	
toggleimageprop	Name of adapter property that provides a URL-string that defines the toggle image. The toggle icon is shown on the right end of a menu item that has subitems.	Optional	
menustyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
menustyleprop	Name of adapter property that dynamically provides explicit style information for the control.	Optional	
withinactivenodes	Set the value of this property to true if you want to disable menu items. In this case, the Natural adapter is generated with an additional INACTIVE field.	Optional	true false

75 DLMENU

Example

The example looks as follows:



A double-line menu is displayed. When selecting a menu item, then its text is written to the status bar.

The XML layout definition is:

```
<page model="menue_02_dl_Adapter">
    <titlebar name="Double Line Menu">
    </titlebar>
    <dlmenu menuprop="menuData">
    </dlmenu>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The DLMENU control is positioned directly following the title bar. In its property `menuprop`, it holds a binding to the property `menuData`.

The Java code of the adapter is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.DLMenu;
import com.softwareag.cis.server.util.DLMenuSubItem;
import com.softwareag.cis.server.util.DLMenuTopItem;

public class menue_02_dl_Adapter
    extends Adapter
{
    // class >MyDLMenuSubItem<
    public class MyDLMenuSubItem extends DLMenuSubItem
    {
        public MyDLMenuSubItem(DLMenuTopItem topItem, String text)
        {
            super(topItem, text);
        }
        public void invoke()
        {
            outputMessage("S",getText() + " was invoked");
        }
    }
    // property >menuData<
    DLMenu m_menuData = new DLMenu();
    public DLMenu getMenuData() { return m_menuData; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        DLMenuTopItem top;
        MyDLMenuSubItem sub;
```

```
        top = new DLMenuTopItem(m_menuData,"File");
        sub = new MyDLMenuSubItem(top,"New...");
        sub = new MyDLMenuSubItem(top,"Save");
        sub = new MyDLMenuSubItem(top,"Save as...");
        sub = new MyDLMenuSubItem(top,"Remove");
        sub = new MyDLMenuSubItem(top,"Exit");
        top = new DLMenuTopItem(m_menuData,"Edit");
        sub = new MyDLMenuSubItem(top,"Undo");
        sub = new MyDLMenuSubItem(top,"Cut");
        sub = new MyDLMenuSubItem(top,"Copy");
        sub = new MyDLMenuSubItem(top,"Paste");
        top = new DLMenuTopItem(m_menuData,"Help");
        sub = new MyDLMenuSubItem(top,"Online Help");
        sub = new MyDLMenuSubItem(top,"About");
    }
}
```

There is an own class `MyDLMenuSubItem` which is subclassed from `DLMenuSubItem` in the package `com.softwareag.cis.server.util`. The main task of this own class is to overwrite the `invoke()` method and to put some logic inside.

Each menu node is represented by an object. Menu nodes of the top line are instances of the class `DLMenuTopItem`. Menu nodes of the second line are instances of the own class `MyDLMenuSubItem`.

All items are arranged inside the member `m_menuData` which is an instance of class `DLMenu`.

When the user clicks an item of the second line at runtime, the `invoke()` method of the corresponding item instance is called in the server.

Properties

Basic			
menuprop	Name of the adapter property that represents the control on server side. The property must be of type "DLMENUInfo". See detailed information inside the Java API Documentation.	Obligatory	
align	Horizontal alignment of the control's content. Default is "center".	Optional	left center right
onlyoneline	If set to "true" then the DLMENU control only contains its top line - there is no second line below. Default is "false".	Optional	true false

cellseparatoronly	If set to "true" then only a very thin cell separator is added between two menu items. Otherwise the separation is rendered explicitly.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
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	

76 Context Menu

The context menu is not a control itself - it is part of existing controls. Context menus are supported by the following controls:

- TEXTGRID2
- TEXTGRIDSSS2
- TREENODE2
- CLIENTTREE
- FIELD

All these controls have items that are represented by corresponding objects on the server side:

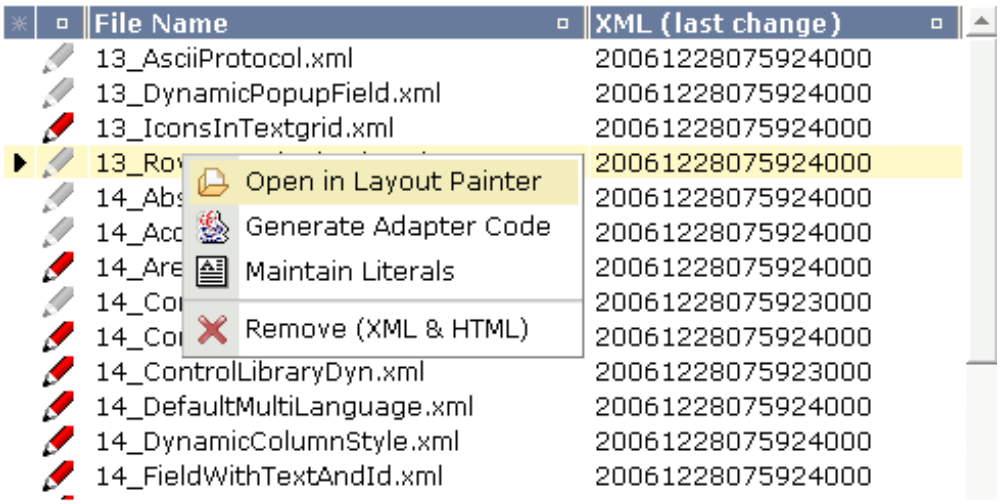
- For the text grid controls, each row of the grid is represented by a certain object on the server side that is managed inside a `TEXTGRIDCollection`.
- For the tree controls, each node of the tree is represented by a certain object (derived from `NODEInfo`) that is managed inside a `TREECollection`.

When the user right-clicks on an item, the method `reactOnContextMenuRequest()` is called inside the item's server-side object. When the user right-clicks in a FIELD control for which "myfield" has been defined with the `valueprop` property, the default name of the called method is `reactOnContextMenuMyfield`. In principle, your server-side implementation could do anything as reaction inside the implementation of the method - i.e. you do not have to open a context menu, but you can also do every other thing that you are able to do inside an adapter implementation.

To open a context menu, you proceed as follows:

- You define a tree collection.
- You add menu nodes to the tree collection - exactly in the same way as you do it with a normal MENU control.
- You pass the tree collection to the adapter via its method `showPopupMenu()`.

Example: Context Menu with a Text Grid



There is no special specification necessary inside the layout definition of the corresponding TEXTGRID2/ TEXTGRIDSSS2 control. All you have to do is to implement the `reactOnContextMenuRequest()` method inside the class representing the items of the grid:

```
public class MenuAdapter
    extends Model
{
    // ----------------------------------------------------------------------
    // inner classes
    // ----------------------------------------------------------------------

    /** class used for pop-up menu. */
    public class MyMenuNodeInfo
        extends MENUNODEInfo
    {
        public MyMenuNodeInfo(String text) { super(text); }
        public MyMenuNodeInfo(String text, String image) { super(text, image); }
        public void reactOnSelect()
        {
           outputMessage("S", "Menu Item \"" + getText() + "\" selected!");
        }
    }

    /** class represents one row within the text grid. */
    public class Line
    {
        String m_name;
        CTimeStamp m_htmlChange;
        CTimeStamp m_xmlChange;
```

```
        public Line(String name, CTimeStamp xmlChange, CTimeStamp htmlChange)
        {
            m_name = name;
            m_xmlChange = xmlChange;
            m_htmlChange = htmlChange;
        }
        public String getName() { return m_name; }
        public CTimeStamp getHtmlChange() { return m_htmlChange; }
        public CTimeStamp getXmlChange() { return m_xmlChange; }

        /** This method will be called if the line will be clicked with
         * the right mouse button.*/
        public void reactOnContextMenuRequest()
        {
            // prepare the appropriate popu menu content
            TREECollection menu = new TREECollection();
            menu.addTopNode(new MyMenuNodeInfo(
                        "Open in Layout Painter",
                        "../HTMLBasedGUI/images/open.gif"),true);
            menu.addTopNode(new MyMenuNodeInfo(
                        "Generate Adapter Code",
                        "../HTMLBasedGUI/images/java.gif"),true);
            menu.addTopNode(new MyMenuNodeInfo(
                        "Maintain Literals",
                        "../HTMLBasedGUI/images/literals.gif"),true);
            menu.addTopNode(new MyMenuNodeInfo(
                        "&SEPARATOR"),true);
            menu.addTopNode(new MyMenuNodeInfo(
                        "Remove (XML & HTML)",
                        "../HTMLBasedGUI/images/remove.gif"),true);

            // open the poup menu
            showPopupMenu(menu);
        }
    }

    ...
    ...
    ...
    ...
    ...

}
```

Pay attention: the `showPopupMenu()` method is provided by the class `Adapter` - in the implementation example, it is directly accessed from the class `Line`. The class `Line` is an inner class and consequently has full access to all the methods of its surrounding class.

Context Menu with a Tree

The implementation of a context menu in the tree is absolutely the same as with a text grid - this time you have to implement the `reactOnContextMenuRequest()` method inside the class representing one tree node.

77 Styling Menus

In many applications you are using different kind of menu controls, for instance context menus and drop down menus. The different menus have common style classes. This allows for applying the same look to different controls. For the styling of the menus, adapt the style variables and/or style classes of your style sheet using the Style Sheet Editor tool. The following provides some hints concerning the style classes shared by several different menu controls as well as the unshared classes.

Shared and Unshared Style Classes in Menu Controls

Both menu controls support separating menu items by a separator. A separator consists of 4 rows:

Row	Description	Style Classes
Row 1	Space above the separator	`MENUSeparatorImgAboveCell`, `MENUSeparatorLabelAboveCell`
Row 2	Separator first line	`MENUSeparatorFirstRow`
Row 3	Separator second line	`MENUSeparatorSecondRow`
Row 4	Space below the separator	`MENUSeparatorImgBelowCell`, `MENUSeparatorLabelBelowCell`

Context and drop down menus use individual style classes and also share some common style classes. They for example share the style classes for the separator mentioned above. The following table lists the most common shared and unshared style classes:

Class	Context Menu	Drop Down Menu	Description
`MENUItemTable`	X	X	Is applied to the `<table>` that encases the menu items.
`MENUItemImageCell`	X	X	Each menu item row consists of an image and a label cell. This style class is applied to the image cell (`<td>` element).
`MENUItemImageCellRollOver`	X	X	When rolling with the mouse over a menu item, this style class replaces the `MENUItemImageCell` style class.
`MENUItemLabelCell`	X	X	Each menu item row consists of an image and a label cell. This style class is applied to the label cell (`<td>` element).
`MENUItemLabelCellRollOver`	X	X	When rolling with the mouse over a menu item, this style class replaces the `MENUItemLabelCell` style class.
`MENUSeparatorImgAboveCell`	X	X	In a separator, this style class is applied to the image (left) cell of the space above (Row 1).
`MENUSeparatorLabelAboveCell`	X	X	In a separator, this style class is applied to the label (right) cell of the space above (Row 1).

Class	Context Menu	Drop Down Menu	Description
MENUSeparatorFirstRow	X	X	In a separator, this style class is applied to the image and label cell of the separator first line (Row 2).
MENUSeparatorSecondRow	X	X	In a separator, this style class is applied to the image and label cell of the separator second line (Row 3).
MENUSeparatorImgBelowCell	X	X	In a separator, this style class is applied to the image (left) cell of the space below (Row 4).
MENUSeparatorLabelBelowCell	X	X	In a separator, this style class is applied to the label (right) cell of the space below (Row 4).
MENUTextCell	X		Use this style class to define the text style of context menu items. The text of a menu item is rendered within an <a> element. This <a> element is embedded in a <td> element. This style class is applied to that <td> element.
MENUTextCellRollOver	X		When rolling the mouse over an item within a context menu, this style class replaces the MENUTextCell style class.
MENUTextCellInactive	X		When a context menu item is set to inactive (i.e. not selectable), this style class is applied to the <td> element, which renders the text.
MENUTop		X	In contrast to context menus, drop down menus have a header line, which is always visible. This header line displays the top nodes. Each top node is rendered within a <table> element. This style class is applied to that <table> element.
MENUTopRollOver		X	When rolling the mouse over a top node, this style class replaces the MENUTop style class.
MENUTopPressed		X	When clicking on a top node, this style class replaces the MENUTopRollOver style class.
MENUTopTextCell		X	Use this style class to define the text within header lines. The text of a top node is rendered within an <a> element. This <a> element is embedded in a <td> element. This style class is applied to that <td> element.
MENUTopTextCellRollOver		X	When clicking on a top node, this style class replaces the MENUTopTextCell style class.
MENUDropDownTextCell		X	Use this style class to define the text style of drop down items other than top nodes. The text of a menu item is rendered within an <a> element. This <a> element is embedded in a <td> element. This style class is applied to that <td> element.

Class	Context Menu	Drop Down Menu	Description
MENUDropDownTextCellRollOver		X	When rolling the mouse over an item within a drop down table, this style class replaces the MENUDropDownTextCell style class.
MENUDropDownTextCellInactive		X	When a menu item other than a top node is set to inactive (i.e. not selectable), this style class replaces the MENUDropDownTextCell style class.

VII Non-Visual Controls and Hot Keys

This part describes some controls that do not have any visual effect to your screen, but provide some client functions to be applied to your page.

The information provided in this part is organized under the following headings:

78 AUTOCOMPLETE

The AUTOCOMPLETE control adds additional functionality to a FIELD control. It supports the selection of a value from a pre-populated list of values while typing. The list of values can be populated from different data sources. On selection, additional values can be displayed in other controls.

General Information

To add autocomplete functionality to a FIELD control, you drag an AUTOCOMPLETE control to your layout page. This AUTOCOMPLETE control can be referenced as data source from within multiple FIELD controls. Adding a reference to an AUTOCOMPLETE data source for a FIELD control is done via the `id` property of the AUTOCOMPLETE control and the `autocompleteref` property of the FIELD control.

```
<autocomplete id="myautocompleteid" ...></autocomplete>
<field valueprop="myinputcontrol" autocompleteref=" myautocompleteid">
</field>
```

In the AUTOCOMPLETE control, you define from which data source the value list will be populated. The `source` property specifies the kind of data source and the `sourcelocation` property a concrete source depending on the kind of data source. You can specify values for the source and source location either at design time or dynamically at runtime. For the latter, the properties `sourceprop` and `sourcelocationprop` are supported.

Populating the value list is done while typing into the FIELD control. With the property `minlength`, you can specify the number of characters after which the value list is to be populated. With the property `delay`, you can specify a delay between a keystroke and the population of the list.

In the example below, the value list is not populated unless at least three characters have been inserted. The population of the list with the values from the data source starts 20 milliseconds after the third character has been inserted.

```
<autocomplete id="myautocompleteid" minlength="3" delay="20" ...></autocomplete>
```

The size of the drop-down box which renders the value list can be customized with the property `rows`. For example, if `rows="5"` is specified, the corresponding drop-down box will have a height of 5 rows. If the value list contains more values, a scrollbar is shown.

Example

Examples which show the usage of the AUTOCOMPLETE control are provided in the Natural for Ajax demos: programs `FDAUTO-P` and `FFAUTO-P`.

Data Sources for Populating the Values

The kind of data source you are using highly depends on the number of total values, whether the values are static or change dynamically on your application context and whether the application is showing internal application data or data from an external service. Also in some applications it is sufficient to populate a simple value list, other applications need to show additional data for each value. The range of supported data sources is from simple static string values to services which deal with thousands of values.

The following kinds of data sources are available.

- String
- File
- Url

String

Choose this data source if you simply want to show a small list of simple values. The values can either be defined at design time or at runtime via the `sourcelocation` property or the `sourcelocationprop` property. In the example below, the total number of values is four. The possible values are defined at design time.

```
<autocomplete id="mycitiesid" source="string"
sourcelocation="Bielefeld;Darmstadt;Frankfurt;Karlsruhe"></autocomplete>
```

When the user types the character "D", all values for which the first character matches "D" are offered by default. In the above example, the value list will contain the item "Darmstadt". We are calling this “match mode”. In addition to this default match mode called "start", another match mode called "all" is supported. When the match mode is "all" and the user types the character "D", the value list will be populated with all items containing a "D". In the above example, these are "Darmstadt" and "Bielefeld". The match mode can be specified via the property `matchmode`. For example:

```
<autocomplete id="mycitiesid" source="string"
sourcelocation="Bielefeld;Darmstadt;Frankfurt;Karlsruhe" ↩
matchmode="all"></autocomplete>
```

File

Choose this data source if you have a medium number of total values and either the offered values are fixed or you have a small number of variations of the total list. The "file" data source is more powerful than the "string" data source because you can show additional data for the offered values.

The data must be stored in a file with simple JSON format (see also *https://www.w3schools.com/js/js_json_intro.asp*) and must be accessible from within the web application. The following formats are supported:

Simple

Example:

```
[
"Bielefeld",
"Darmstadt",
"Frankfurt",
"Karlsruhe"
]
```

The offered functionality is the same as for String (see above). The advantage is that the values are neither in the page layout nor in the Natural code. This allows for simple replacement of the files when the offered values change without touching any application code.

Value

Example:

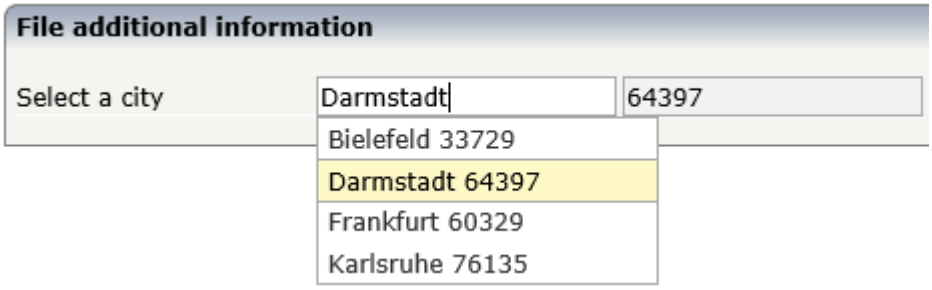
```
[
{ "code": "33729", "value": "Bielefeld" },
{ "code": "64397", "value": "Darmstadt" },
{ "code": "60329", "value": "Frankfurt" },
{ "code": "76135", "value": "Karlsruhe" }
]
```

Each offered item has a "value" for populating the value list and can have additional data like "code" as in the above example. The same match modes are supported as for String (see above). Choose this format if you want to show additional data for the offered values.

Showing additional data is done by using the properties `autocompletedisplayname` and `autocompletedisplayref` in the FIELD control. The following example shows how to use "code" as an additional value in a second FIELD control:

```
<autocomplete id="mycity" source="file"
sourcelocation="./autocomplete/mycities.txt"></autocomplete>
<itr>
     <label name="Select a city"></label>
     <field valueprop="fldcity" autocompleteref="mycity"
      autocompletedisplayname="code" autocompletedisplayref="fldcode"></field>
     <hdist></hdist>
    <field valueprop="fldcode" displayonly="true"></field>
</itr>
```

The additional data will automatically be shown in the drop-down box.



The check for matching items is done on both "value" and "code". This means, when the user types "6", the list will be populated in the following way:

If you do not like this special handling for the additional data, choose the “Value and Label” format described below.

Value and Label

Example:

```
[
{ "label":"33729 Bielefeld", "value":"Bielefeld", "code":"33729" },
{ "label":"64397 Darmstadt", "value":"Darmstadt", "code":"64397" },
{ "label":"60329 Frankfurt", "value":"Frankfurt", "code":"60329" },
{ "label":"76135 Karlsruhe", "value":"Karlsruhe", "code":"76135" }
]
```

The "label" is shown in the drop-down box and the "value" is taken over to the FIELD control. The check for matching items is done on the "label" as a whole. Typing "D" with `matchmode="start"` will result in no items.

Additional data can be specified in the FIELD control using the properties `autocompleteref` and `autocompletedisplayname` as described above for the “Value” format.

Url

Choose this data source if the data is provided by a service. This is usually the case when the number of total values is very high and/or the values are read from databases like Adabas. This is the most flexible data source. Other than with the data sources String and File, the service is in charge of finding the matching values. The service will only return the values which match the characters typed in by the end-user. The service is free to implement any match mode it likes; therefore, the `matchmode` property is not supported.

The service must return the data in a simple JSON format. The same formats as described for the data source File are supported. The characters that the user types in are passed as request parameter "q" to the service.

Example: When the user types the characters "Da", the service *http://myhost/MyService* will be called as follows: *http://myhost/MyService?q=Da*.

A service can be implemented as a simple Java Servlet. With Natural RPC and EntireX, a service can also be written in Natural. The NJXDEMOS contain an example Natural subprogram, corresponding EntireX wrapper classes and configuration settings. See the document *NaturalAutocompleteRPCService.pdf* in the subfolder *njxdemos\autocomplete* of the NaturalAjaxDemos example project.

Properties

Basic			
id	The id of the autocomplete data source. This id can be referenced from FIELD controls.	Obligatory	
source	The kind of data source like string, file, url.	Optional	string file url
sourcelocation	The source location. Depends on the kind of source. Examples: ./autocomplete/myfile, http://myremotedatasource...	Optional	
minlength	The drop down selection box is not opened before a minlength number of characters is inserted into the FIELD control.	Optional	1 2 3 int-value
delay	The delay in milliseconds between when a keystroke occurs and when a search is performed	Optional	1

			2 3 int-value
matchmode	Specify "start" to find all items which start with the typed in characters. Specify "all" to find all items which contain the typed in characters. Default is "start". For source="url", the matchmode is ignored.	Optional	all start
rows	Height of the drop down box in rows. If more items are found, a scrollbar is shown.	Optional	1 2 3 int-value
maxresults	The maximum number of results displayed in the dropdown box.	Optional	1 2 3 int-value
Binding			
sourcelocationprop	Name of the adapter property that dynamically sets the sourcelocation at runtime.	Optional	
sourceprop	Name of the adapter property that provides the kind of source dynamically at runtime.	Optional	

79 TIMER

With a timer, you can regularly trigger a defined method invoked by the client. For example, you can use a timer to regularly update information to be displayed inside your page.

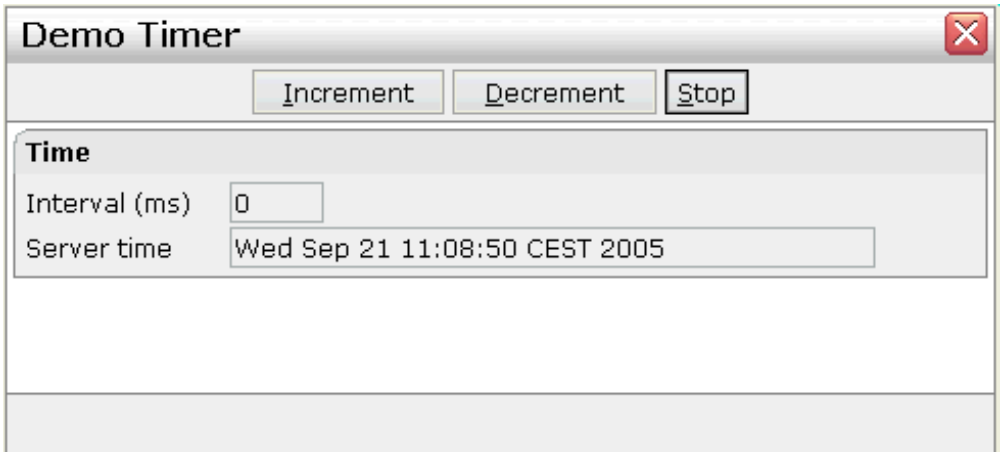
The timer tag is accessible as a valid subnode inside the page tag.

Specify either the `interval` or the `intervalprop` property in order to set the interval. In case of using a property for dynamically setting the interval, note the following:

- You can change the interval time at any time.
- You can stop the timer by setting the interval time to 0.

Example

The following screen displays a time stamp of the server. It is refreshed depending on the interval field. Increase/decrease the interval time by choosing the corresponding buttons.



The XML layout definition is:

```
<page model="DemoTimerAdapter">
    <titlebar name="Demo Timer">
    </titlebar>
    <header withdistance="false">
        <button name="~~Increment" method="incrementTimer">
        </button>
        <button name="~~Decrement" method="decrementTimer">
        </button>
        <button name="~~Stop" method="stopTimer">
        </button>
    </header>
    <pagebody>
        <rowarea name="Time">
            <itr>
```

```
                <label name="Interval (ms)" width="100" asplaintext="true">
                </label>
                <field valueprop="interval" length="5" displayonly="true" ↩
datatype="int">
                </field>
            </itr>
            <itr>
                <label name="Server time" width="100" asplaintext="true">
                </label>
                <field valueprop="serverTime" length="50" displayonly="true">
                </field>
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
    <timer intervalprop="interval">
    </timer>
</page>
```

In this example, the timer tag does not call a defined method but refreshes the screen. The timer interval is retrieved by the property `interval` of the adapter object.

The Java code of the adapter is:

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

import java.util.Date;
import com.softwareag.cis.server.Adapter;

public class DemoTimerAdapter
    extends Adapter
{
    // property >interval<
    int m_interval=1000;
    public int getInterval() { return m_interval; }
    public void setInterval(int value) { m_interval = value; }

    // property >serverTime<
    String m_serverTime;
    public String getServerTime()
    {
        Date d = new Date();
        return d.toString();
    }

    public void decrementTimer() { m_interval -= 500; }
    public void incrementTimer() { m_interval += 500; }
    public void stopTimer() { m_interval = 0; }
}
```

It just contains the property `serverTime`, returning the current time and the property `interval` to provide the interval duration, controlling the timer. The `stopTimer` method sets the interval duration to "0".

Tip: Do not update the page too frequently. Every update means a roundtrip to the server.

Properties

Basic			
interval	Duration in milliseconds the timer waits between calling the adapter method defined in the METHOD property. Use this property to "hard code" the duration - or use INTERVALPROP to define the duration by an adapter property.	Sometimes obligatory	
intervalprop	Name of adapter property that defines the timer interval's duration. The adapter property must be of type "int" or "Integer" ("long"/ "Long"/ "String"). If "0" is passed then the timer is stopped.	Sometimes obligatory	
method	Name of adapter method that is called by the timer.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

80 Extended Hot Key Management

Extended hot key management provides the following features:

- Possibility to define hot keys with certain controls.
- Possibility to define language dependent hot keys.

Direct Hot Key Definitions with Certain Controls

Some controls allow to directly specify hot keys within the text that is displayed inside the control. The controls that currently support this feature are:

- BUTTON
- MENU
- ROWTABAREA

Example: If you specify the button text to be "~~Stop", the button will look like this:

The text may both be directly maintained in the control (`name` property) or may come from the multi language management (`textid` property).

At the time, the hot key CTRL+ALT+S will be added to the page. The definition of hot keys in the texts of MENU controls or ROWTABAREA controls is done in the same way.

Caution: Application Designer does not check if hot keys are defined twice in a page.

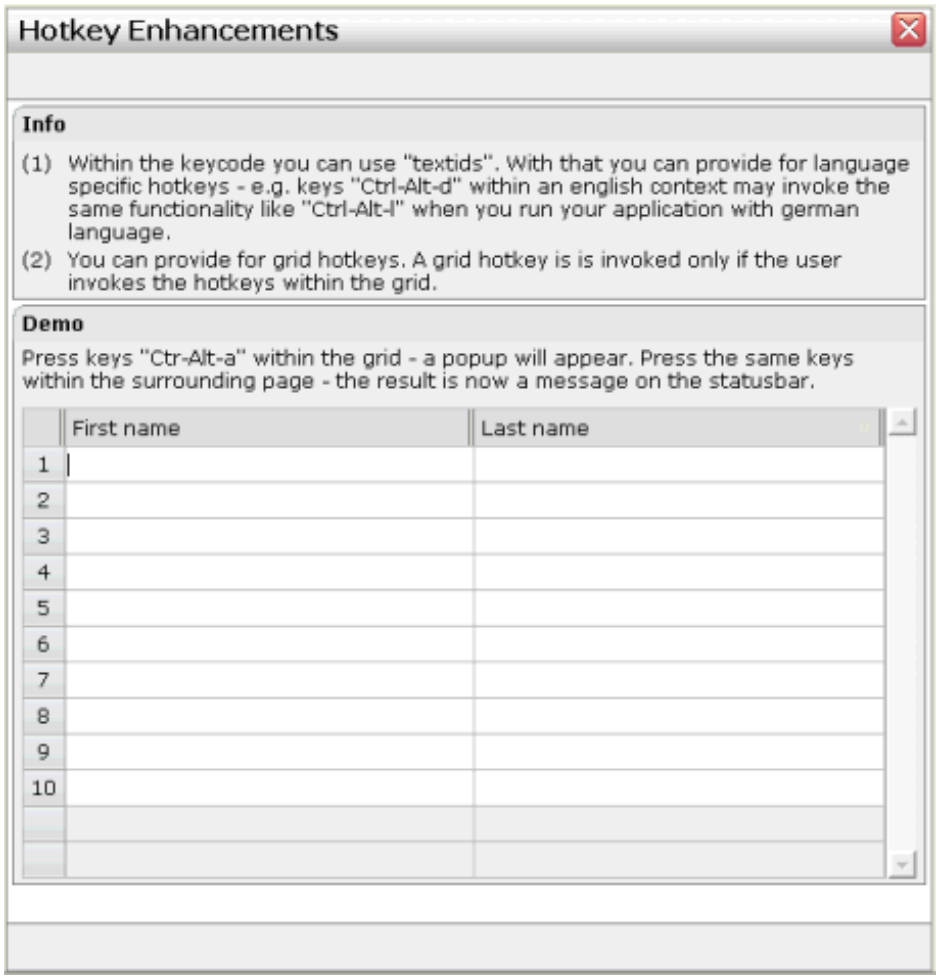
Why use CTRL+ALT as a default way to trigger the hot keys? This is because most of the simple ALT keys are already occupied by the browser.

Hot Key Definitions for Certain Controls

The controls PAGE, FIELD and ROWTABLEAREA2 support the property `hotkeys`.

The `hotkeys` property defines the active hot keys for the corresponding control. This means that you may have hot keys that are only valid inside a certain grid (ROWTABLEAREA2 control) or even inside a single FIELD, but are not valid inside the whole page (PAGE control).

Have a look at the following demo:



If the user presses CTRL+ALT+A inside the grid, the hot key is managed by the grid. If the user presses the same key outside the grid, the hot key is processed by a corresponding definition on page level. The XML layout looks as follows:

```
<page model="com.softwareag.cis.test40.GridHotkeysAdapter" ↩
translationreference="40_gridhotkeys"
      hotkeys="ctrl-alt-65;onCtrlAltAPage">
...
...
        <rowtablearea2 griddataprop="grid" rowcount="12" width="100%" ↩
firstrowcolwidths="true"
                      hotkeys="ctrl-alt-$KEYCODE_A;onCtrlAltA">
...
...
```

The `hotkeys` property on PAGE, FIELD or ROWTABLEAREA2 is a semicolon-separated list containing the hot key itself and the method it is calling. There can be multiple hot key definitions for the same control. When maintaining this property, use the special dialog in the Layout Painter that appears for the `hotkeys` property. For further information, see *Defining Hot Keys* in the *Development Workplace* documentation.

You can either specify the key code of the hot key or a text ID that is to be translated by the multi language management.

VIII Controls for Absolute Positioning

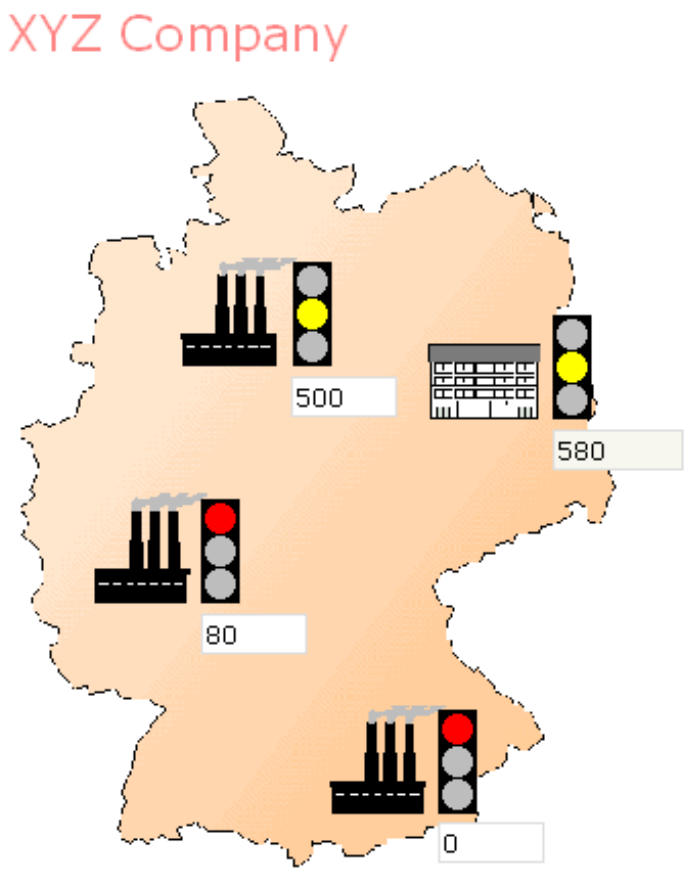
There is a special set of controls available to position them by defining their absolute x- and y-co-ordinates. In addition, you can define the z-coordinate to define the drawing sequence if two controls overlap.

Use these controls for specific purposes only - positioning controls by using container controls is much more simple and much more flexible.

The information provided in this part is organized under the following headings:

81 Example

A typical case for using absolute positioning is demonstrated in the following example:



On top of a map there are normal input fields and dynamic icons. If you enter certain values, the traffic lights change their color.

The controls supporting absolute positioning start with the prefix "ABS".

This is the XML code for the above example:

```
<page model="com.softwareag.cis.demo.AbsoluteDemoAdapter">
    <absfolder name="All">
        <absfolder name="Center">
            <absdynicon valueprop="imgCenter" x="522" y="264" z="10">
            </absdynicon>
            <absfield valueprop="kfCenter" length="10" x="522" y="319" z="10" ↩
displayonly="true">
            </absfield>
        </absfolder>
        <absfolder name="Factory1">
            <absdynicon valueprop="imgFactory1" x="332" y="225" z="10">
            </absdynicon>
            <absfield valueprop="kfFactory1" length="10" x="332" y="280" z="10">
            </absfield>
        </absfolder>
        <absfolder name="Factory2">
            <absdynicon valueprop="imgFactory2" x="270" y="396" z="10">
            </absdynicon>
            <absfield valueprop="kfFactory2" length="10" x="270" y="451" z="10">
            </absfield>
        </absfolder>
        <absfolder name="Factory3">
            <absdynicon valueprop="imgFactory3" x="440" y="549" z="10">
            </absdynicon>
            <absfield valueprop="kfFactory3" length="10" x="440" y="604" z="10">
            </absfield>
        </absfolder>
        <absicon image="images/absbackground.gif" x="100" y="70" z="1">
        </absicon>
        <abslabel x="20" y="80" z="10" name="XYZ Company" textsize="5" ↩
textcolor="#FF8080">
        </abslabel>
    </absfolder>
    <titlebar name="Overview">
    </titlebar>
    <header>
        <button name="Refresh" method="refresh">
        </button>
    </header>
    <pagebody vscroll="false">
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

Some controls start with "ABS" and hold x-, y- and z-coordinates. These controls are structured within ABSFOLDER controls. The ABSFOLDER controls have neither optical or execution relevance

- they are just used to structure the absolutely positioned controls inside a hierarchy - otherwise, the controls would be displayed in one long line, one after the other.

The Java adapter code looks as follows:

```
package com.softwareag.cis.demo;

import com.softwareag.cis.server.Adapter;

public class AbsoluteDemoAdapter extends Adapter
{
    float m_kfFactory1;
    float m_kfFactory2;
    float m_kfFactory3;
    float m_kfCenter;

    public void setKfFactory1(float value) { m_kfFactory1 = value; }
    public float getKfFactory1() { return m_kfFactory1; }

    public void setKfFactory2(float value) { m_kfFactory2 = value; }
    public float getKfFactory2() { return m_kfFactory2; }

    public void setKfFactory3(float value) { m_kfFactory3 = value; }
    public float getKfFactory3() { return m_kfFactory3; }

    public float getKfCenter() { return m_kfCenter; }

    public void reactOnDataTransferEnd()
    {
        m_kfCenter = m_kfFactory1 + m_kfFactory2 + m_kfFactory3;
    }

    public String getImgFactory1() { return findImage(m_kfFactory1); }
    public String getImgFactory2() { return findImage(m_kfFactory2); }
    public String getImgFactory3() { return findImage(m_kfFactory3); }
    public String getImgCenter() { return findImage(m_kfCenter/3); }

    private String findImage(float f)
    {
        if (f < 1000) return "images/abstlred.gif";
        if (f < 10000) return "images/abstlyellow.gif";
        return "images/abstlgreen.gif";
    }
}
```

Properties starting with "kf" represent key figures which are displayed in the page. All properties starting with "img" pass back a name of an image file. The image file is used inside the ABS-DYNICON control in the layout description to show an image which is taken from the value of an adapter property.

82 Controls

All controls which allow absolute positioning have 3 standard properties:

- x - this is the X coordinate.
- y - this is the Y coordinate.
- z - this is the Z coordinate.

The x- and y-value is measured from the top left corner of the page.

The z-value indicates the drawing sequence - the layer. The layer information becomes important if controls overlap - the control with the higher z-value is drawn on top of the control with the lower z-value.

83 ABSFOLDER

The ABSFOLDER control has neither optical or execution relevance. It is used to structure the absolutely positioned controls inside a hierarchy. Without the ABSFOLDER control, the controls would be displayed in one long line, one after the other.

Properties

Basic			
name	A name for the ABSFOLDER can be defined here, without any effect on rendering and behaviour.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

84 ABSFIELD

The ABSFIELD is the normal FIELD control to be positioned absolutely. All properties are the same as the properties used by the FIELD control. See the description of the FIELD control.

Properties

Basic			
valueprop	Server side property representation of the control.	Obligatory	
length	Width of FIELD in amount of characters. WIDTH and LENGTH should not be used together. Note that the actual size of the control depends on the font definition if using the LENGTH property.	Optional	5 10 15 20 int-value
x	X-coordinated (in pixels) of the left top corner of the control.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the control.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the control.	Optional	
flush	Flushing behaviour of the input control. By default an input into the control is registered within the browser client - and communicated to the server adapter object when a user e.g. presses a button. By using the FLUSH property you can change this behaviour. Setting FLUSH to "server" means that directly after changeing the input a synchronization with the server adapter is triggered. As consequence you directly can react inside your adapter logic onto the change of the corresponding value. - Please be aware of that during the synchronization always all changed properties - also the ones that were changed before - are transferred to the adapter object, not only the one that triggered the synchonization. Setting FLUSH to "screen" means that the changed value is populated inside the page. You	Optional	screen server

	use this option if you have redundant usage of the same property inside one page and if you want to pass one changed value to all its representation directly after changing the value.		
flushmethod	When the data synchronization of the control is set to FLUSH="server" then you can specify an explicit method to be called when the user updates the content of the control. By doing so you can distinguish on the server side from which control the flush of data was triggered.	Optional	
password	If set to "true", each entered character is displayed as a '*'.	Optional	true false
displayonly	If set to true, the FIELD will not be accessible for input. It is just used as an output field.	Optional	true false
displayprop	Name of adapter property that controls whether the field is displayonly(true) or not (false). By using this property you can dynamically control the "display"-status of the control by your adapter object.	Optional	
statusprop	Name of the adapter property that dynamically passes information how the field should be rendered and how it should act.	Optional	
popupmethod	Name of the adapter's method that is called when the user requests value help by pressing F4 or F7 or by clicking into the FIELD with the right mouse button. See at chapter 'Popup Dialog Management' for more details. If the POPUPMETHOD is defined, a small icon is shown inside the field to indicate to the user that there is some value help available.	Optional	openIdValueCombo openIdValueHelp openIdValueComboOrPopup
datatype	By default, the FIELD control is managing its content as string. By explicitly setting a datatype you can define that the control... ...will check the user input if it reflects the datatype. E.g. if the user inputs "abc" into a field with datatype "int" then a corresponding error message will popup when the user leaves the field. ...will format the data coming from the server or coming form the user input: if the field has datatype "date" and the user inputs "010304"	Optional	date float int long time timestamp color

	then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). In addition value popups are offered for the user automatically for some datatypes: e.g. when specifying datatype "date" the automatically the field provides a calendar input popup. Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.		xs:decimal xs:double xs:date xs:dateTime xs:time ---------- N n.n P n.n string n L xs:boolean xs:byte xs:short
fieldstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

85 ABSICON

This is an image which is drawn at a defined coordinate. Either show the image in the original size or define the image size explicitly. Optionally, define a method to be called when the image is clicked.

Properties

Basic			
image	Name of the image to be displayed as an icon. The value must be a valid URL.	Obligatory	
method	Name of the adapter method to be called by clicking on the icon. If the name is not specified, the data of the page is synchronized with the server by clicking on the icon.	Obligatory	
x	X-coordinated (in pixels) of the left top corner of the control.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the control.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the control.	Optional	
title	Title of the icon to be displayed as a "tool tip". If the mouse cursor stays on the icon for some time, the title will appear.	Optional	
titletextid	Text id of the icon's title, replaced by a literal by the multi language management at runtime.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

86 ABSDYNICON

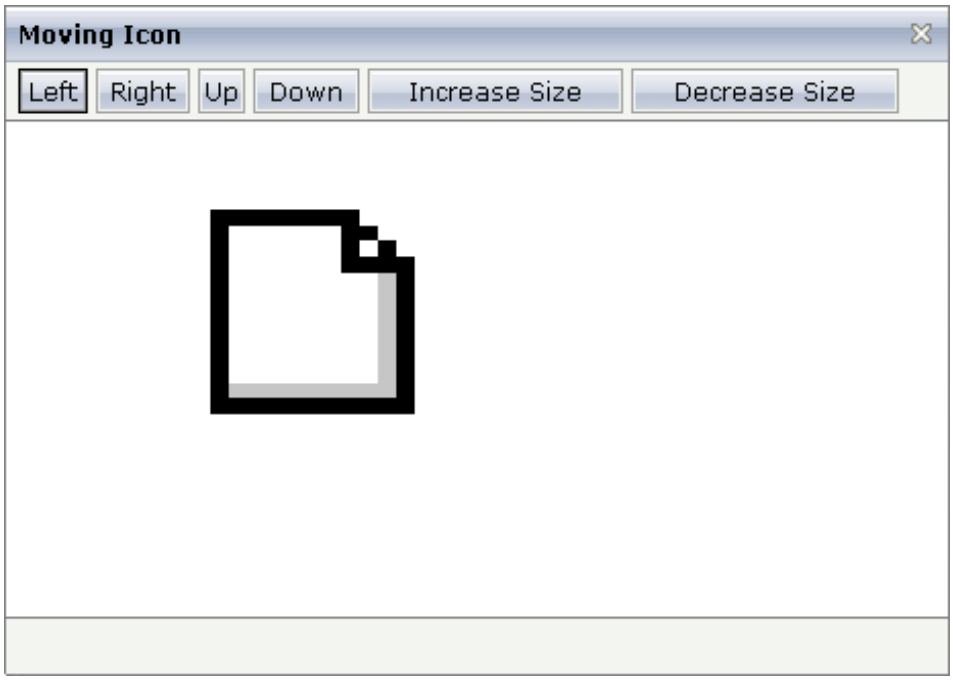
The ABSDYNICON is similar to the **ABSICON** control. Main difference: whereas the name of the image is defined statically by the ABSICON control, it is derived from an adapter property by the ABSDYNICON control. The image can be changed by the application. The ABSDYNICON is used to display the dynamic traffic lights in the **previous** example.

It is also possible to specify the height and the width of an ABSDYNICON control by binding these values to corresponding adapter properties. For example, use this feature to display a numeric value by a graphical bar which changes its size depending on this value.

In addition, it is also possible to specify the coordinates (`x`/`y`/`z`) of the ABSDYNICON dynamically. Maybe you want to display an icon representing a car and want to update regularly its position by properties of your adapter object.

Example: Moving an Icon

The following example demonstrates the possibility to specify dynamically the size and coordinates of the ABDSDYNICON control. It looks as follows:



By clicking on the buttons you manipulate the size and the position of the icon.

The XML layout definition looks as follows:

```
<page model="MovingIconAdapter">
    <absfolder name="ABSFolderMovingIcon">
        <absdynicon valueprop="iconName" xprop="x" yprop="y" zprop="z" ↩
heightprop="height"
                    widthprop="width">
        </absdynicon>
    </absfolder>
    <titlebar name="Moving Icon">
    </titlebar>
    <header align="left" withdistance="false">
        <button name="Left" method="moveLeft">
        </button>
        <button name="Right" method="moveRight">
        </button>
        <button name="Up" method="moveUp">
        </button>
        <button name="Down" method="moveDown">
        </button>
        <button name="Increase Size" method="increase">
        </button>
        <button name="Decrease Size" method="decrease">
        </button>
    </header>
    <pagebody>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The coordinates and the size for the ABSDYNICON control are derived from adapter properties. The adapter class source is:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class MovingIconAdapter
    extends Adapter
{
    // property >height<
    int m_height=100;
    public int getHeight() { return m_height; }
    public void setHeight(int value) { m_height = value; }

    // property >iconName<
    String m_iconName= "images/new.gif";
    public String getIconName() { return m_iconName; }
    public void setIconName(String value) { m_iconName = value; }

    // property >width<
    int m_width=100;
```

```
    public int getWidth() { return m_width; }
    public void setWidth(int value) { m_width = value; }

    // property >x<
    int m_x=100;
    public int getX() { return m_x; }
    public void setX(int value) { m_x = value; }

    // property >y<
    int m_y=100;
    public int getY() { return m_y; }
    public void setY(int value) { m_y = value; }

    // property >z<
    int m_z;
    public int getZ() { return m_z; }
    public void setZ(int value) { m_z = value; }

    /** */
    public void decrease()
    {
        m_width -= 20;
        m_height -= 20;
    }

    /** */
    public void increase()
    {
        m_width += 20;
        m_height += 20;
    }

    /** */
    public void moveLeft() { m_x -= 20; }
    public void moveRight() { m_x += 20; }
    public void moveUp() { m_y -= 20; }
    public void moveDown() { m_y += 20; }
}
```

Properties

Basic			
valueprop	Name of the adapter property providing the URL of the image to be displayed as an icon.	Obligatory	
method	Method being called inside the adapter when clicking on the icon. You do not have to define a value but can also use the icon just as a dynamic image display.	Optional	
x	X position in pixels.	Optional	
y	Y position in pixels.	Optional	
z	Z position.	Optional	
xprop	Name of property returning the X position.	Optional	
yprop	Name of property returning the Y position.	Optional	
zprop	Name of property returning the Z position.	Optional	
height	Height of icon in pixels.	Optional	
width	Width of icon in pixels.	Optional	
heightprop	Name of property returning the height.	Optional	
widthprop	Name of property returning the width.	Optional	
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	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

There are three options to set the size of an icon:

1. The icon is displayed in its original size - you do not have to specify any of the properties `height`, `width`, `heightprop`, `widthprop`.
2. The icon is displayed with a defined size which does not change - you have to specify the `height` and `width` property values with the corresponding pixel values.
3. The icon is displayed with a defined size which changes dynamically - you have to specify the `heightprop` and `widthprop` property values and you have to provide the corresponding adapter properties.

There are two options for setting the position of an icon:

1. Static definition by the `x`, `y` and `z` properties.
2. Dynamic definition by adapter properties which are specified by the `xprop`, `yprop` and `zprop` properties.

87 ABSTEXTOUT

The ABSTEXTOUT control allows you to display text information which is dynamically derived from an adapter property.

Properties

Basic			
x	X-coordinated (in pixels) of the left top corner of the control.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the control.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the control.	Optional	
valueprop	Name of the adapter property providing the text to be displayed.	Obligatory	
textsize	The HTML font size of the text. Corresponding to the HTML definition "1" means "smallest" and "6" means "biggest".	Optional	1 2 3 4 5 6
textcolor	Colour in which the text is displayed. Must be a valid colour code, e.g. #FF0000 for "red".	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
datatype	By default, the control is managing its content as string. By explicitly setting a datatype you can define that the control will format the data coming from the server: if the field has datatype "date" and the user inputs "010304" then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings). Please note: the datatype "float" is named a bit misleading - it represents any decimal format number. The server side representation may be a float value, but also can be a double or a BigDecimal property.	Optional	date float int long time timestamp color

			xs:decimal xs:double xs:date xs:dateTime xs:time ---------- N n.n P n.n string n L xs:boolean xs:byte xs:short
textoutstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

88 ABSLABEL

The ABSLABEL allows you to display static text information. The text can be a text which is taken from the multi language management.

Properties

Basic			
x	X-coordinated (in pixels) of the left top corner of the area.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the area.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the area.	Optional	
name	Static text which is displayed.	Optional	
textid	Text id, replaced by a literal of the multi language management.	Optional	
textsize	Font size of the text as defined by the HTML font size specification.	Optional	1 2 3 4 5 6
textcolor	Colour in which the text is displayed.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
labelstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon.	Optional	

	Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Either use the `name` or the `textid` property.

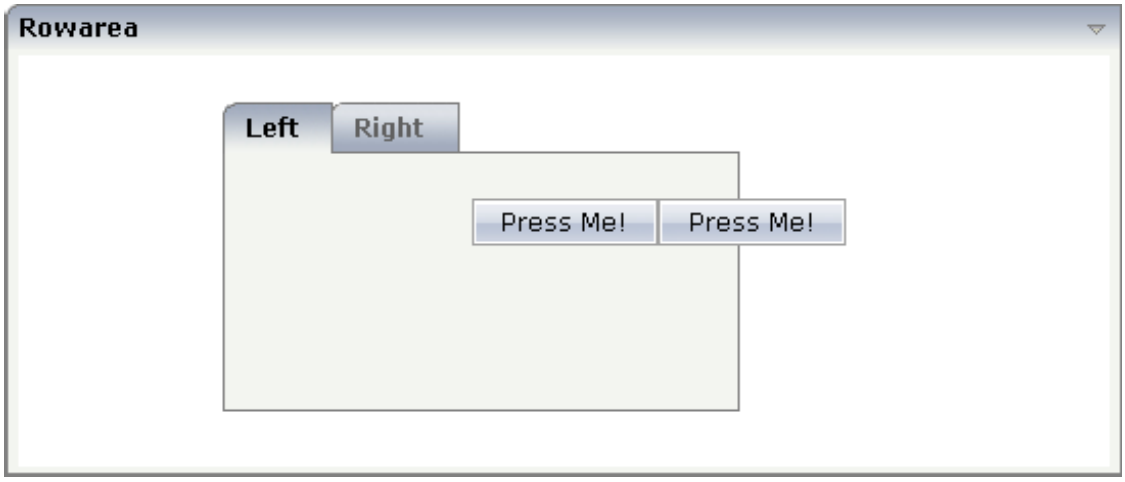
89 ABSTABLE0/ABSTR

This is a set of very powerful absolute controls because they are very generic in what happens below.

The ABSTABLE0 represents a table that you place with its top left corner onto a certain x, y, z position. Inside the table, you can do what you want - i.e. you can include any other controls reachable inside the table.

The ABSTR represents a row that you place in the same way as ABSTABLE0.

This is an example:



The XML layout definition is:

```
<rowarea name="Rowarea">
    <rowabsarea width="100%" height="200">
        <abstable0 x="100" y="20" z="1" width="250">
            <rowtabarea height="150" name1="Left" page1="idPage1"
                        name2="Right" page2="idPage2">
                <tabpage id="Left" takefullheight="true">
                </tabpage>
                <tabpage id="Right" takefullheight="true">
                </tabpage>
            </rowtabarea>
        </abstable0>
        <abstr x="220" y="70" z="2">
            <button name="Press Me!" method="onPressMe">
            </button>
            <button name="Press Me!" method="onPressMe">
            </button>
        </abstr>
    </rowabsarea>
</rowarea>
```

ABSTABLE0 Properties

Basic			
x	X-coordinated (in pixels) of the left top corner of the area.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the area.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the area.	Optional	
xprop	Name of adapter properties for the x-coordinate.	Optional	
yprop	Name of adapter properties for the y-coordinate.	Optional	
zprop	Name of adapter properties for the z-coordinate.	Optional	
height	Height in pixels. Only required if you use percentage sizing inside the ABSTABLE0. Otherwise the height of the table follows the height of its content.	Optional	100 150 200 250 300 250 400 50% 100%
width	Width in pixels. If not defined then ABSTABLE0 will be as wide as required by its content.	Optional	100 120 140 160 180 200 50% 100%
takefullheight	Indicates if the content of the control's area gets the full available height. If you use percentage sizing inside the control's area then this property must be switched to 'true'. If you use no explicit	Optional	true false

	vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'. Background information: container control's internally open up a table in which you place rows (ITR/TR) which then hold controls (e.g. LABEL/FIELD). The table that is opened up normally has no explicit height and grows with its content as consequence. By specifying "takefullheight=true" the table itself is sized to fill the maximum height of the available area.		
fixlayout	The fixlayout property is important for saving rendering performance inside your browser. To become effective it requires to have specified the height and the width (if available as property) of the control. If setting fixlayout to "true" then the control's area is defined as area which is not sized dependent on its content (as normally done with table rendering). Instead the size is predefined from outside without letting the browser "look" into the content of the area. If the content is not fitting into the area then it is cut. You typically use this control if the content of the control's area is flexibly sizable. E.g. if the content (e.g. a TEXTGRID control) is following the size of the container. When using vertical percentage based sizing you should pay attention to set the fixlayout-property to "true" as often as possible. - The browser as consequence will be much faster in doing its rendering because a screen consists out of "building blocks" with simple to calculate sizes.	Optional	true false
tablestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

ABSTR Properties

Basic			
x	X-coordinated (in pixels) of the left top corner of the area.	Optional	
y	Y-coordinated (in pixels) of the left top corner of the area.	Optional	
z	Z-coordinated (in pixels) of the left top corner of the area.	Optional	
xprop	Name of adapter properties for the x-coordinates.	Optional	
yprop	Name of adapter properties for the y-coordinates.	Optional	
zprop	Name of adapter properties for the z-coordinates.	Optional	
visibleprop	Name of an adapter property that provides the information if this control is displayed or not. As consequence you can control the visibility of the control dynamically. The server side property needs to be of type "boolean".	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

90 ROWABSAREA

It is possible to define flexible areas in which absolutely positioned controls are displayed. There are three possibilities:

- You define all controls directly inside the page tag.
- You define the page body to hold the controls.
- You define explicitly the area for control positioning inside any part of the page.

All Controls Directly Inside the Page Tag

This is the easiest way to use absolutely positioned controls - and it is used in the **example** previously in this part. Each coordinate of a control relates to the whole generated HTML page, i.e. the coordinate "(0;0)" positions a control at the left-top corner of the page.

Though this is the fastest way to start with, there are some disadvantages:

- If a control is positioned outside the page, the user is unable to scroll to this point.
- If you add controls which are not absolutely positioned, you have to mix both variants in a dangerous way: e.g. define a title bar and a header inside a page. Therefore, you will not position your controls on top of these elements - i.e. you will position them at a y-coordinate "60" in order to keep a distance. If further elements are added inside the header, the height of the header increases and all absolutely positioned controls need to be redefined by a new y-coordinate.

All Controls Inside the Page Body

The page body (PAGEBODY tag) typically reflects the area between header and status bar. It is possible to add a ROWABSAREA control inside the page body. Inside the ROWABSAREA container, you can position your controls.

If you do not define any further parameters by this ROWABSAREA control, it will occupy the whole page body if necessary. Defining the `vscroll` and `hscroll` properties (set to true) inside the PAGEBODY tag, you can scroll to any control that is outside the visible range of the page body.

Look at the following example: it contains two absolutely positioned icons inside the page body. If the window size is too small to show both icons, the scroll bars are shown accordingly.

The layout definition of this example looks as follows:

```
<pagebody vscroll="true" hscroll="true">
    <rowabsarea>
        <absfolder>
            <absicon image="images/logo.gif" x="150" y="100" z="10">
            </absicon>
            <absicon image="images/logo.gif" x="250" y="200" z="10">
            </absicon>
        </absfolder>
```

```
    </rowabsarea>
</pagebody>
```

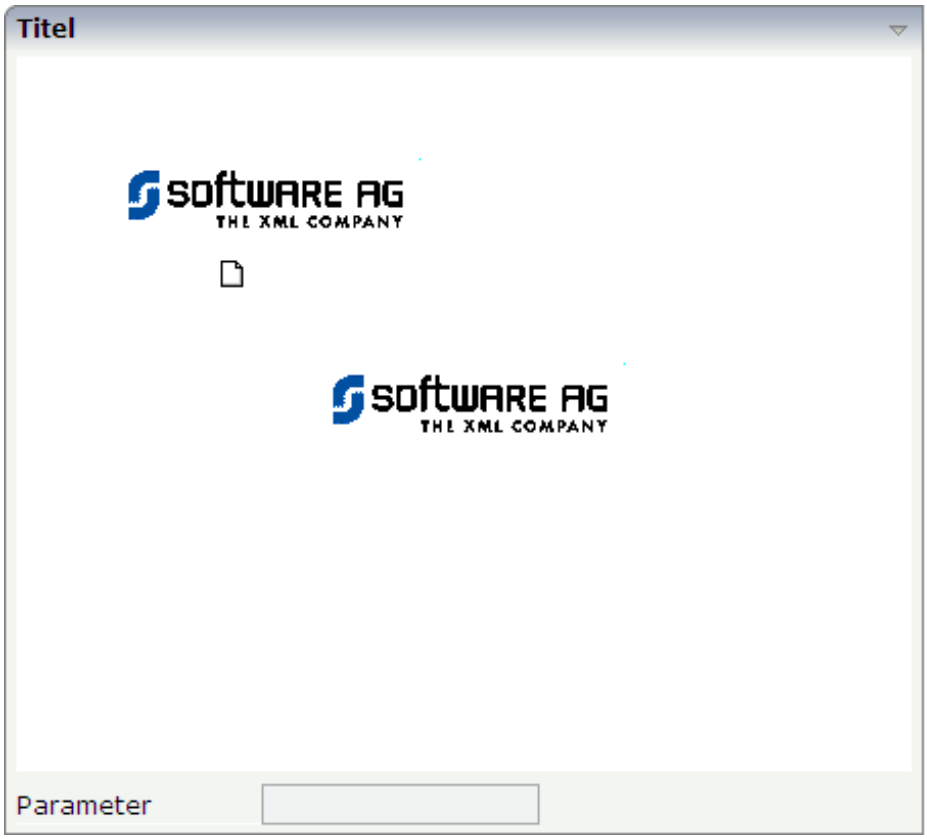
The controls are positioned relative to the page body's coordinates. I.e. the coordinate "(0;0)" is the left-top corner of the page body.

Use this method to position the controls within a page where the page body is used for positioned controls absolutely.

Explicit Areas for Absolute Positioning

Use the ROWABSAREA container, which is explained above, to define areas inside a page where you want to position controls.

Have a look at the following example:



An area was embedded for absolute positioning inside the normal flow of controls of a page. The corresponding XML layout definition looks as follows:

```
<rowarea name="Titel">
    <rowabsarea width="100%" height="350">
        <absfolder>
            <absicon image="images/logo.gif" x="50" y="50" z="100">
            </absicon>
            <absicon image="images/new.gif" x="120" y="120" z="100">
            </absicon>
            <absicon image="images/logo.gif" x="150" y="150" z="100">
            </absicon>
        </absfolder>
    </rowabsarea>
    <vdist height="5">
    </vdist>
    <itr>
        <label name="Parameter" width="120" asplaintext="false">
        </label>
        <field valueprop="factor1" length="20">
        </field>
    </itr>
</rowarea>
```

Inside the ROWAREA definition, a ROWABSAREA is placed - holding a defined width and size. Below the ROWABSAREA, the normal definition of a set of absolutely positioned controls are defined, i.e. an ABSFOLDER with some ABSICON definitions.

In the row following the ROWABSAREA, normal controls are defined - just as usual.

Properties

Basic			
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 percentage 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%
height	Height of the control.	Optional	100

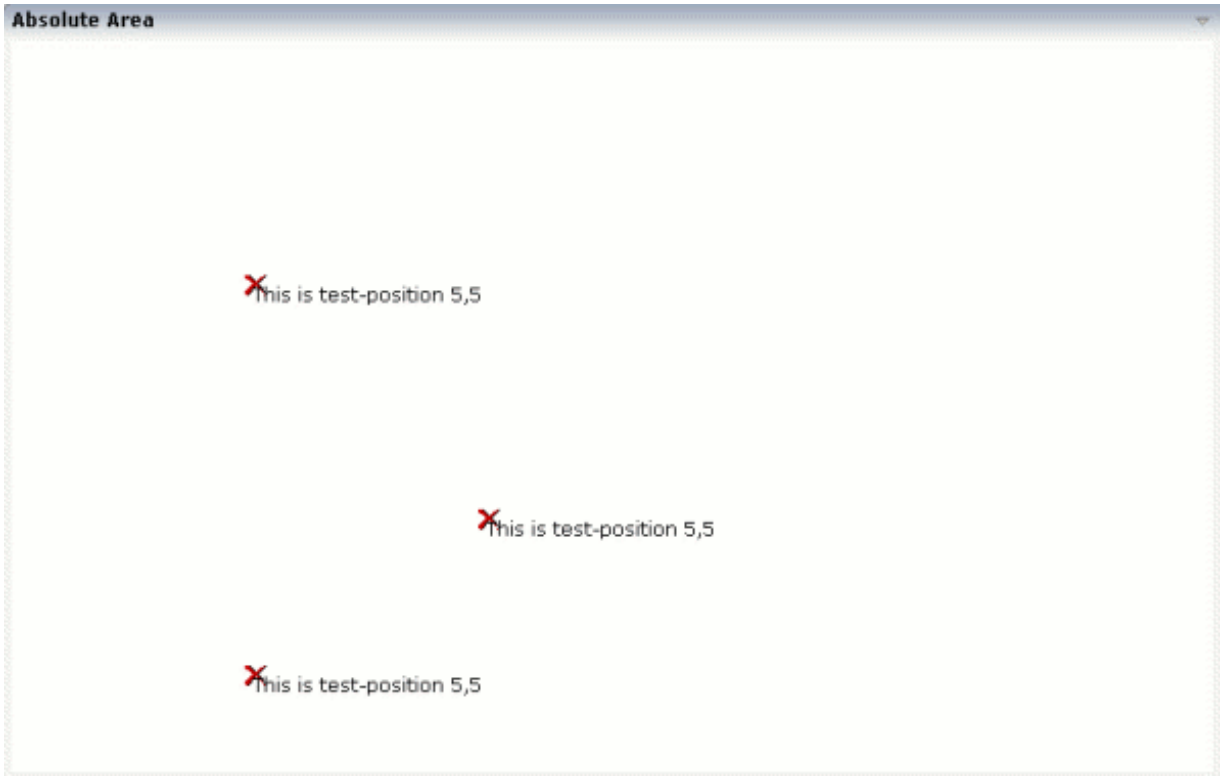
	There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

91 ABSAREA

Independent of what type of area you define for positioning controls (see previous section), it is possible to define subareas inside this area. A subarea is an area with its own x-, y- and z-coordinates and also contains absolutely positioned controls - or again: subareas.

Example

The following example shows a page with three groups of absolutely positioned controls - each group holding an icon (ABSICON) and a label (ABSLABEL):



The XML layout definition contains three definitions of an ABSAREA control - with different x-, y- and z-coordinates. Inside the area, the controls are positioned with exactly the same coordinates.

```
<rowarea name="Absolute Area" height="500">
    <rowabsarea width="100%" height="100%">
        <absarea x="150" y="150" z="10">
            <abslabel x="5" y="5" z="10" name="This is test-position 5,5">
            </abslabel>
            <absicon image="images/remove.gif" x="0" y="0" z="0">
            </absicon>
        </absarea>
        <absarea x="150" y="400" z="10">
            <abslabel x="5" y="5" z="10" name="This is test-position 5,5">
            </abslabel>
```

```
            <absicon image="images/remove.gif" x="0" y="0" z="0">
            </absicon>
        </absarea>
        <absarea x="300" y="300" z="10">
            <abslabel x="5" y="5" z="10" name="This is test-position 5,5">
            </abslabel>
            <absicon image="images/remove.gif" x="0" y="0" z="0">
            </absicon>
        </absarea>
    </rowabsarea>
</rowarea>
```

An ABSAREA control opens its own area on the page, providing its own coordinate system.

In this example, the x-, y- and z-coordinates for each area are defined inside the layout definition. Set the position dynamically by deriving the x-, y- and z-values by the adapter properties.

Properties

Basic			
x	X-coordinate (in pixels) of the left top corner of the area.	Optional	
y	Y-coordinate (in pixels) of the left top corner of the area.	Optional	
z	Z-coordinate (in pixels) of the left top corner of the area.	Optional	
xprop	Name of adapter properties for the x-coordinates.	Optional	
yprop	Name of adapter properties for the y-coordinates.	Optional	
zprop	Name of adapter properties for the z-coordinates.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Either all of the properties `x`, `y` and `z` have to be defined or all the properties `xprop`, `yprop` and `zprop` have to be defined.

IX Working with Pages

This documentation deals with more complex applications in which you have sequences of pages. The information is subdivided into the following parts:

Working with Page Navigation	Describes how to develop a sequence for page navigation.
Embedding Pages into Pages	Describes how to develop a master page and embed other pages within it.
Multi Frame Pages	Describes how to generate a HTML frameset page.
Embedding Pages into a Workplace	Describes how to integrate pages into portal or workplace environments.

92 Working with Page Navigation

In more complex applications, you often have to cope with situations in which you have to go through a sequence of pages. For example, for entering a purchase order, you have to specify first some header information (like customer, address, etc.), go to a list of items you want to order, open detail information page(s) on a selected item, etc.

The navigation can be quite complex on its own - there are several frameworks available which deal with this topic.

What Application Designer offers is a way to navigate between different pages. How you find out when and where to navigate to (server-side business logic) - is not of interest for Application Designer. As soon as you know where to go to, tell Application Designer your decision.

The following topics are covered below:

Page Navigation

Page navigation is triggered by the adapter class. Typically, it is a method call which is triggered by a button click. The `Adapter` class from which you derive your adapter class, offers some methods which make page navigation very simple.

- The First Navigation
- Preparing the Adapter before Navigating
- Including the Adapter while Navigating

The First Navigation

The most simple way of navigating can be seen in the following code example:

```
public void showNextPage()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Please first input all fields");
        return;
    }
    // open new page
    this.switchToPage("pageName.html");
}
```

In this method, first there is a check whether navigation is “appropriate” in the current situation. If not, an error message is shown in the status bar. Otherwise, navigation is done by the inherited method `switchToPage(pageName)`.

Preparing the Adapter before Navigating

In our example, the next screen is - as usual - linked with a specific adapter class. An instance of this adapter class is generated by the session management.

If you want to prepare the adapter of the next screen in a certain way, you proceed as follows.

Before navigating to the next page, you can ask for the adapter which is linked to the next page:

```
public void showNextPage()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
       ...
        this.outputMessage("E","Please first input all fields");
        return;
    }
    // prepare adapter object which corresponds to next page
    XYZModel m = (XYZModel)this.findAdapter(XYZModel.class);
    m.setParam1(...);
    m.setParam2(...);
    ...
    ...
    // open new page
    this.switchToPage("pageName.html");
}
```

The method `findAdapter` returns the adapter object which is assigned to the next page. Therefore, you are able to prepare the adapter by setting any information you want to show in the next screen.

Including the Adapter while Navigating

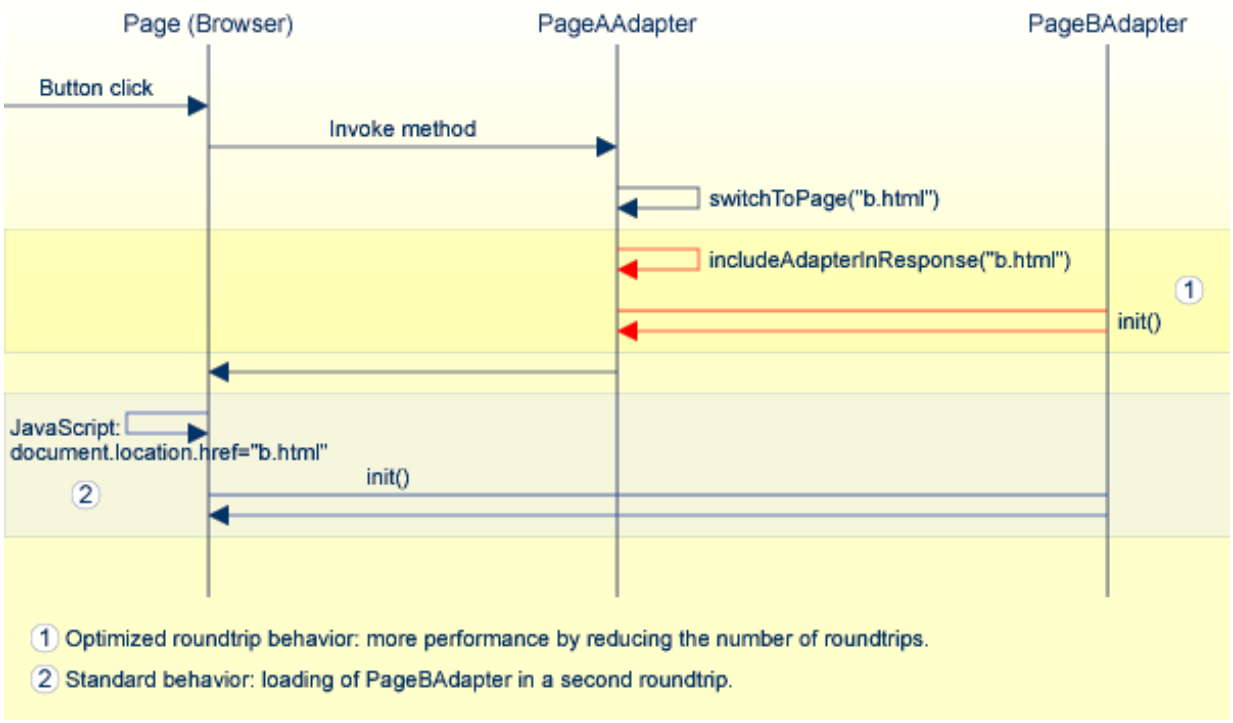
The following example shows how to increase performance for page navigation using the method `includeAdapterInResponse`. It is typically used with:

- `switchToPage();`
- `openPopup();`
- `openCISPageInTarget()`

Example:

```
public void showNextPage()
{
    // open new page
    switchToPage("pageName.html");
    // initialize the corresponding Adapter of pageName.html
    includeAdapterInResponse("pageName.html", true);
}
```

The method `includeAdapterInResponse` includes the adapter of the second page (page B) into the response processing of the first page (page A). The adapter is processed in the same way as it is processed when being called by an explicit HTTP request coming from the browser. This is an effective mechanism for reducing the number of roundtrips between the browser and the server (it reduces the number of roundtrips from two to one). This is illustrated by the following diagram:



Note: In a local area network (LAN) environment, the gain in performance will not be significant. However, in a slow wide area network (WAN) environment, the performance will be improved significantly.

Session Management

You might ask: who controls the life cycle of the adapter classes? If I navigate from page "A" to page "B" and go back to page "A": do I come back to the adapter object I was already using, or do I get a new adapter instance?

Session, Subsession, Adapter

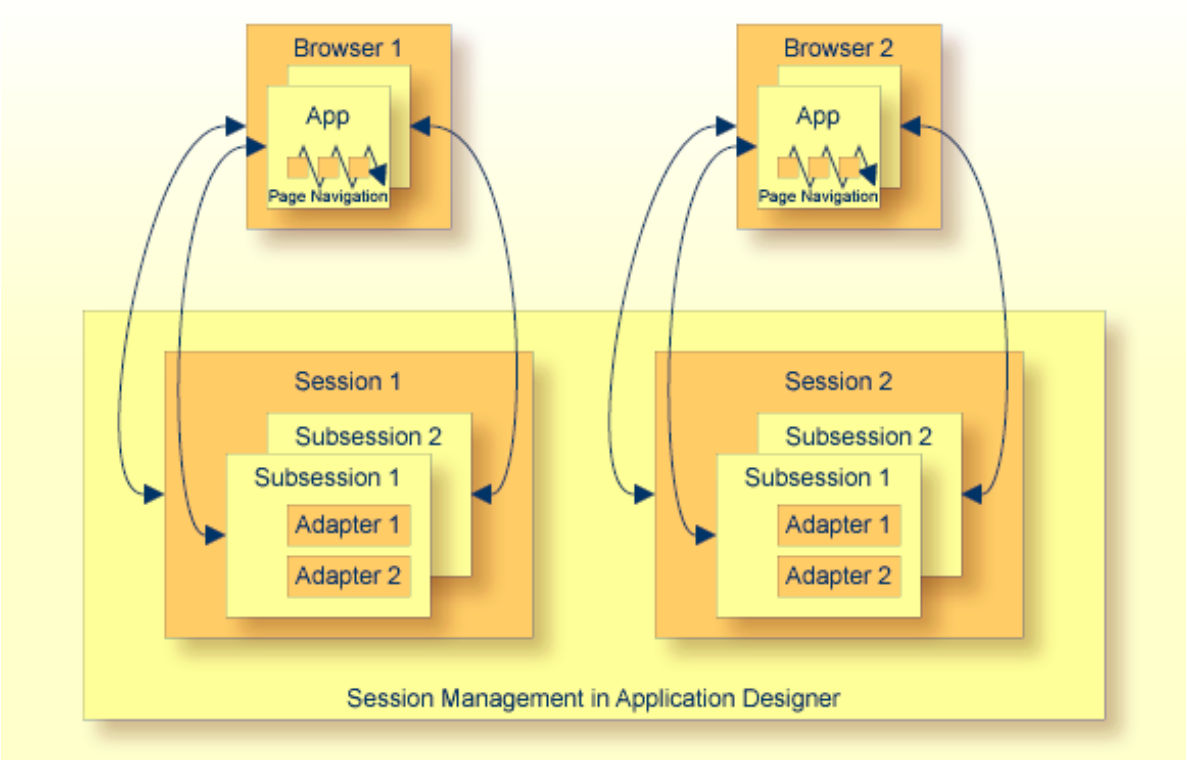
The management of the adapters inside the server is done by the session management of Application Designer. Typically you do not have to take care of it - it is done automatically in front of your adapters.

Every browser instance connected to Application Designer creates a session and is assigned to it at the server side. If you start another browser instance, a second session is created internally which is completely decoupled from all other sessions. And so on.

A session is divided into subsessions. A subsession is a logical separation of independent activities which run parallel within the context of one session. Example: in the workplace, you can run various applications in parallel. You can switch from one application to the other. Each running application is represented by an instance of a subsession at the server side. The subsessions are also completely isolated from each other.

Within a subsession, the adapter instances are held. The basic rules for managing these instances inside one subsession are:

- For each adapter class one instance is kept. This means: if a page requests an adapter, it is first determined whether this adapter instance is already created within the subsession. If yes, the existing instance is used, otherwise a new adapter instance is created and registered.
- The adapter instance is held for the whole life cycle of the subsession - as long as not explicitly removed by the adapter logic.
- All variant and page navigation is done inside a subsession as described in this section.



Page navigation within the browser is a navigation between adapter instances of the same subsession.

Garbage Collection

The final garbage collection of adapter instances is done by removing a subsession - if not explicitly controlled in a different way by the adapter logic. The adapter class offers the method `endProcess()` which removes the subsession you are just working with:

```
public void exit()
{
     // check if you really want to exit
     if ( ... )
     {
          ...
          this.outputMessage("E","Cannot exit due to...");
          return;
     }
     // exit
     this.endProcess();
}
```

Whenever a user logs off, the session - including all subsessions and its assigned adapter instances - is removed from the session management and released for garbage collection.

Opening Modal Pop-up Dialogs

Pop-up dialogs are just normal Application Designer pages (except for a small difference) which are opened in modal pop-up mode. The pop-up management does not start a new browser instance - everything is done in the same instance in which you are working.

Invoking a pop-up dialog follows the same rules as navigating between pages. The Java source of the adapter looks as follows:

```
public void showPopup()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Opening pop-up is not possible...");
        return;
    }
    // open new page
    this.openPopup("pageName.html");
}
```

The adapter - which is used as a server-side counterpart of the pop-up dialog - is managed like navigating between pages. Therefore, you can access the adapter before opening the pop-up dialog and prepare some content:

```
public void showPopup()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Opening pop-up is not possible...");
        return;
    }
    // prepare adapter object which corresponds to next page
    XYZModel m = (XYZModel)this.findAdapter(XYZModel.class);
    m.setParam1(...);
    m.setParam2(...);
    ...
    ...
    // open new page
    this.openPopup("pageName.html");
}
```

- Special Pop-up Dialog Parameters within the XML Layout Definition
- Passing Pop-up Dialog Parameters before Opening a Pop-up
- Closing a Pop-up Dialog

- Changing the Size within an Opened Pop-up

Special Pop-up Dialog Parameters within the XML Layout Definition

Any Application Designer page can be opened as a pop-up dialog. Inside the PAGE tag of the page, you can define how to open the pop-up dialog. There are a couple of properties which can be used for this purpose:

- `popupwidth`
- `popupheight`
- `popupfeature`

For further information, see the PAGE property definition in the *Java Page Layout* > *Typical Page Layout* documentation.

Passing Pop-up Dialog Parameters before Opening a Pop-up

The pop-up parameters (width, height, features) can also be passed before calling a pop-up. The `Adapter` class offers corresponding interfaces. The following code shows how to open a pop-up with a certain title and with a certain size and position:

```
/** */
public void onOpenPopup()
{
    setPopupFeatures(100, // x-position
                      100, // y-position
                      300, // width
                      200, // height
                     ""   // additional features as string (see PAGE-POPUPFEATURES ↩
docu)
                      );
    setPopupTitle("This is the title of the pop-up");
    openPopup("25_PositionedPopup1.html");
}
```

The parameters you pass override the parameters that may be defined in the pop-up page's layout definition.

Closing a Pop-up Dialog

A pop-up dialog can be closed by its corresponding adapter by the `closePage()` method which is inherited from the `Adapter` class:

```
/** This method is bound to the exit button of the pop-up page. */
public void exitPopup()
{
    // check if can be closed
    ...
    ...
    // close pop-up
    this.closePage();
}
```

In addition, a user can always close a dialog by pressing ALT+F4 or by choosing the close icon at the top right corner of the window title. The adapter - both adapters, the pop-up adapter and the adapter of the page from which the pop-up dialog was called - are not informed about this action and so it always has to be taken into consideration that a pop-up dialog might be closed by the user.

Changing the Size within an Opened Pop-up

Sometimes you need to resize the pop-up in which the user is currently working. For example, you want to show additional information and therefore have to increase the height of the pop-up.

The following code is inside the adapter object that belongs to the opened pop-up page:

```
public void onXxxxxx()
{
    findFunctionsLivingPopup().setPopupSize(m_newWidth,m_newHeight);
}
```

URL to Choose when Navigating

By the `switchToPage(...)` and the `openPopup(...)` methods, a URL is passed as a string parameter to Application Designer for navigation. How can the URL be defined?

You can use relative links as long as the page to which you navigate is in the same directory as the page from which you navigate. This is especially important when navigating between pages which belong to the same application project. See also *Application Project Management* in *First Steps Java Pages > Some Background Information*.

If you want to navigate outside your project, you have to specify a link starting with the document root of your HTTP server. For each application project, a new context path is set up with the name

of the project. Navigating from one project's file to another can be done by specifying the full URL like /<*project*>/<*projectfile.html*>.

Pages created without the project management (such as the Hello World example) are accessible by the default context /*HTMLBasedGUI*/.

Value Help Pop-up Dialogs

In case you want to provide help in form of a pop-up dialog - based on a certain field - Application Designer offers a technique for implementing a value selection help:

- The FIELD control offers the property `popupmethod`. With this property, a method of the adapter class is called whenever the user requests a value help inside the field - by pressing F4 or F7 in the field or by double-clicking on the field.
- A pop-up dialog opens displaying possible data selections.

The value help pop-up dialogs are just normal pop-up dialogs which just have a dedicated purpose.

Standard Method openIdValueHelp

Inherited from the `Adapter` class, there is a very simple way to provide a value help pop-up dialog. The method `openIdValueHelp` is implemented in a generic way and can be used as follows:

- In your adapter, implement a method `findValidValuesForXxx()`. Replace "Xxx" with the name of the property field.
- This method must return an array of `com.softwareag.cis.server.util.ValidValueLine` objects. This array contains pairs of IDs and values which are valid data options for the `Xxx` property.
- When requesting a value help for the corresponding field, a pop-up dialog displays the `ValidValueLine` objects which are passed back from your method. If the user selects an item in the pop-up dialog, the value is placed in the `setXXX` method of the property.

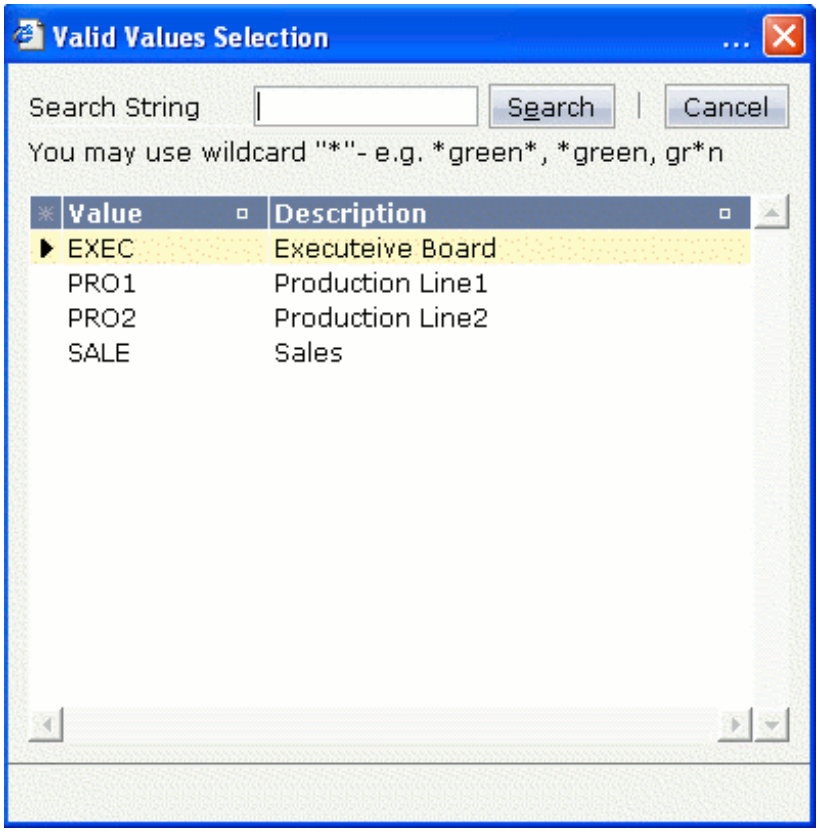
The following Java source shows an example:

```
// property >department<
   String m_department;
   public String getDepartment() { return m_department; }
   public void setDepartment(String value) { m_department = value; }

   public ValidValueLine[] findValidValuesForDepartment()
   {
       Vector v = new Vector();
       v.addElement(new ValidValueLine("EXEC","Executive Board"));
       v.addElement(new ValidValueLine("PRO1","Production Line1"));
       v.addElement(new ValidValueLine("PRO2","Production Line2"));
       v.addElement(new ValidValueLine("SALE","Sales"));
       ValidValueLine[] result = new ValidValueLine[v.size()];
       v.copyInto(result);
       return result;
   }
```

The XML layout looks as follows:

```
<rowarea name="Field with Value Help">
    <itr>
        <label name="Department" width="100">
        </label>
        <field valueprop="department" width="150" popupmethod="openIdValueHelp">
        </field>
    </itr>
</rowarea>
```

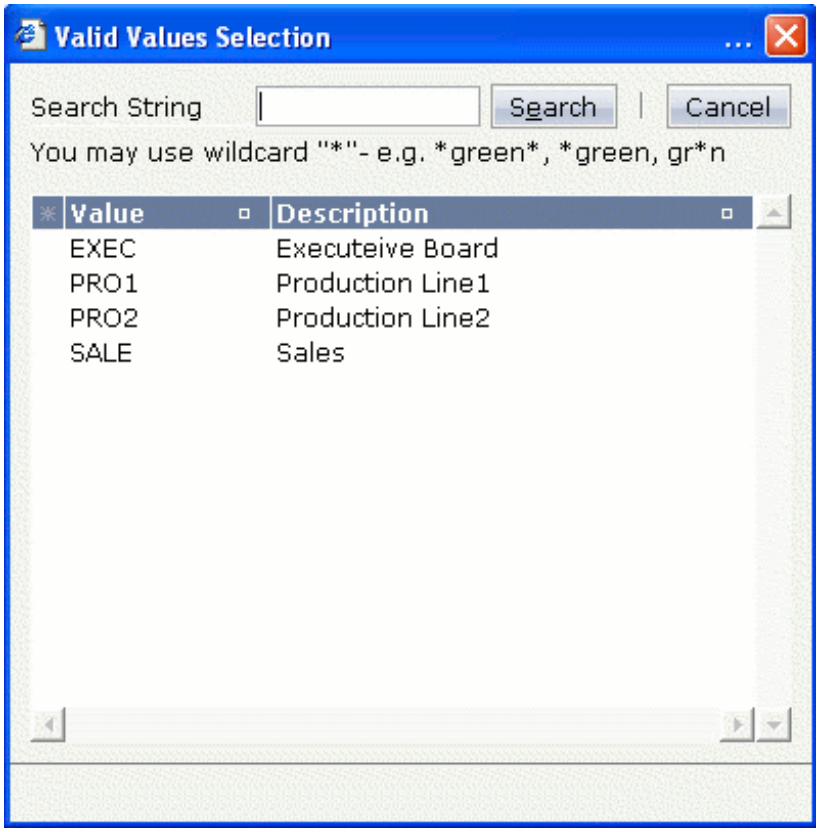
The result is a field which automatically opens a pop-up dialog when the user presses F4 or F7, or double-clicks on the field.



An additional feature available: instead of displaying pairs of ID and name, the dialog can display a list of IDs only. There is a constructor of the `ValidValueLine` class with only passing an ID to it:

```
public ValidValueLine[] findValidValuesForDepartment_02()
    {
        Vector v = new Vector();
        v.addElement(new ValidValueLine("EXEC"));
        v.addElement(new ValidValueLine("PRO1"));
        v.addElement(new ValidValueLine("PRO2"));
        v.addElement(new ValidValueLine("SALE"));
        ValidValueLine[] result = new ValidValueLine[v.size()];
        v.copyInto(result);
        return result;
    }
```

Now the pop-up dialog contains only a column containing the IDs:



Standard Method openIdValueCombo

See the description of the FIELD control for information on how to implement a valid value help with the `openIdValueCombo` method. This method does not open a pop-up but it open a combo-like selection.

The interface on the server side is exactly the same as for `openIdValueHelp` - just the rendering result is different:

Further information is provided in the description of the FIELD control.

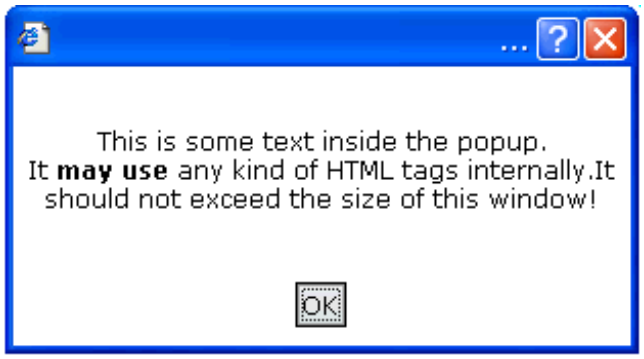
Standard Pop-up Dialogs

There are standard pop-up dialogs available for general usage which you do not have to code yourself.

OK Pop-up

The OK pop-up is used for displaying a text with an **OK** button.

The following is an example of an OK pop-up:



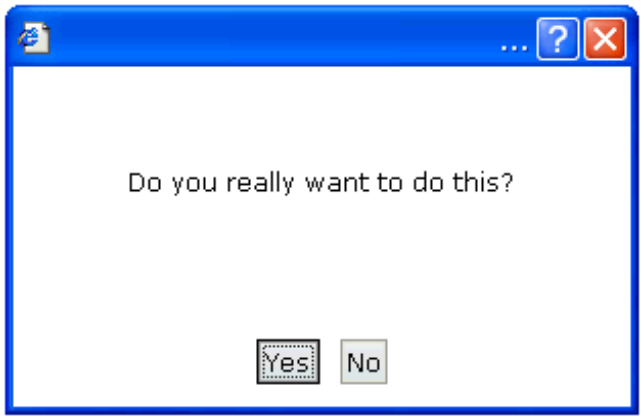
The code of the adapter is:

```
public void showOKPopup()
{
    PopupOKModel pok = (PopupOKModel)findAdapter(PopupOKModel.class);
    pok.init("This is some text inside the pop-up.<br>"+
             "It <b>may use</b> any kind of HTML tags internally." +
             "It should not exceed the size of this window!");
    this.openPopup("/HTMLBasedGUI/popupok.html");
}
```

Yes/No Pop-up

The Yes/No pop-up is used for asking the user a question. Depending on user's decision, activities are started inside the adapter.

The following is an example of a Yes/No pop-up:



The code of the adapter is:

```
public class YESCommand implements com.softwareag.cis.server.util.ICommand
{
    public void execute()
    { outputMessage("S","Yes command was called"); }
}
public class NOCommand implements com.softwareag.cis.server.util.ICommand
{
    public void execute()
    { outputMessage("S","No command was called"); }
}
public void showYESNOPopup()
{
    PopupYesNoModel pyn = (PopupYesNoModel)findAdapter(PopupYesNoModel.class);
    pyn.init("Do you really want to do this?",
              new YESCommand(),
              new NOCommand());
    this.openPopup("/HTMLBasedGUI/popupyesno.html");
}
```

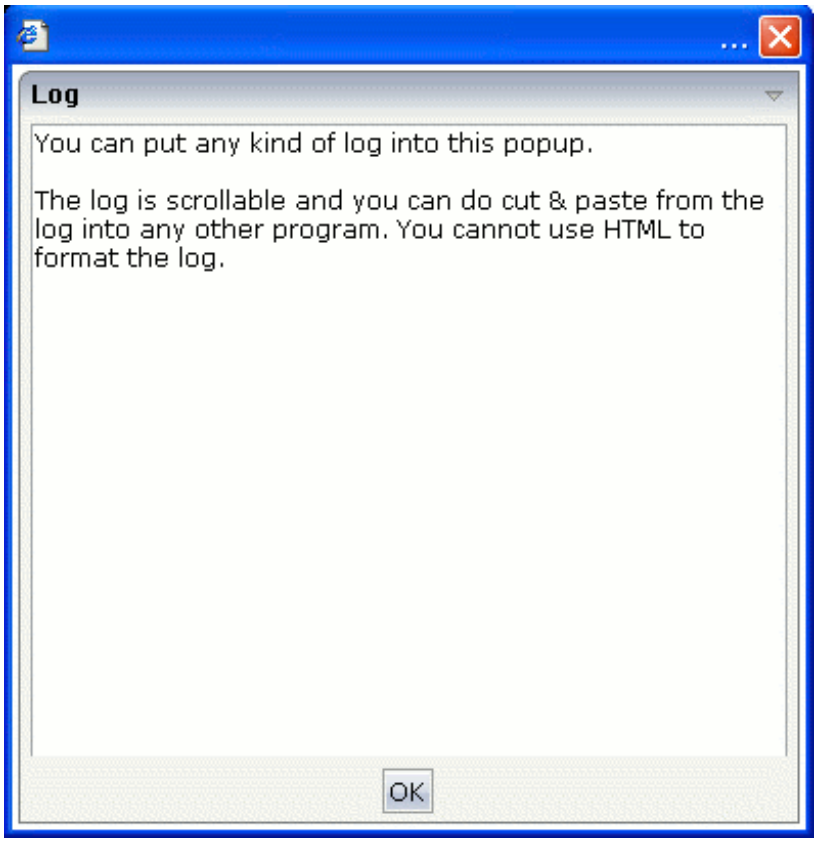
The pop-up dialog is initialised by passing the question and two "reaction objects" to it. One "reaction object" is called when choosing the **Yes** button, the other is called when choosing the **No** button.

The "reaction objects" have to implement the interface `com.softwareag.cis.server.util.Icommand` which just needs a simple `execute()` method. In our example, the "reaction objects" are implemented as inner classes of the adapter class.

Log Pop-up

The Log pop-up is used for displaying a log text.

The following is an example of a Log pop-up:



The code inside the adapter is:

```
public void showLOGPopup()
{
    PopupLogModel plm = (PopupLogModel)findAdapter(PopupLogModel.class);
    plm.init("You can put any kind of log into this pop-up.\n\n"+
             "The log is scrollable and you can do cut & paste from "+
             "the log into any other program. You cannot use HTML "+
             "to format the log.");
    this.openPopup("/HTMLBasedGUI/popuplog.html");
}
```

Example: Asking Whether the User Really Wants to Quit

This is a typical example: the user works on a page of your application for a while and then choose the close icon in the right top corner of the page. Check whether the user has changed something and ask using a pop-up dialog if the user really wants to close the page.

The following Java source shows an implementation in the adapter class:

```
/** */
public void endProcess()
{
    if (m_changed == true)
    {
        PopupYesNoModel pyn = (PopupYesNoModel)findAdapter(PopupYesNoModel.class);
        pyn.init("You modified some data. Do you really want to exit?",
                 new ICommand() { public void execute() { executeEndProcess(); }},
                 null);
        this.openPopup("/HTMLBasedGUI/popupyesno.html");
    }
    else
    {
        executeEndProcess();
    }
}

/** */
public void executeEndProcess()
{
    super.endProcess();
}
```

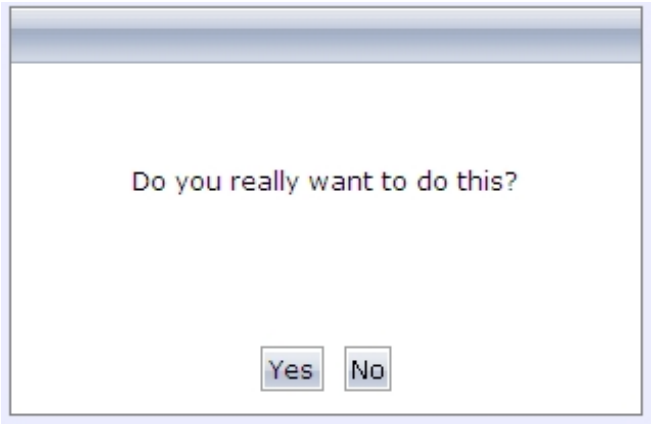
The `endProcess()` method is called by the closing function of the page. It is provided by the `Adapter` class from which the adapter is inherited. The `endProcess()` method does already everything which is required for removing the subsession.

Overwrite the `endProcess()` method and embed the code which opens a Yes/No pop-up to ask whether the user really wants to quit the application. The original closing function is shifted to the method `executeEndProcess()`. The Yes/No pop-up got for the “Yes” method an inner class pointing to the `executeEndProcess()` method. The “No” method is null and means that nothing should be done.

Page-based Pop-up Dialogs

Page-based pop-up dialogs look and behave different from the **standard pop-up dialogs**. The content of a page-based pop-up dialog is not opened within a modal browser window. This has the following advantages:

- Page-based pop-up dialogs are faster than the standard pop-up dialogs.
- There are no browser window drawbacks. A page-based pop-up dialog does not have a close icon; the user must always choose a command button before the page-based pop-up dialog is closed. There is no status bar in which an URL can be shown.
- Page-based pop-up dialogs are not affected by pop-up blockers.
- Page-based pop-up dialogs are page-modal only. This means that they do not block the whole browser window. Other pages can be opened in a subpage or in an other frame.



To open standard pop-up dialogs and page-based pop-up dialogs, you have to use the following API calls:

- Open a standard pop-up dialog in a browser window:

```
openPopup(page)
```

- Open a page-based pop-up dialog:

```
openPagePopup(page)
```

93 Embedding Pages into Pages

In Application Designer, there is the possibility to embed pages into other pages. Or vice versa: you can create pages which consist of other embedded pages. In this case, the (master) page and the embedding page operate independently from each other by having a channel to cooperate.

Use this technique to build pages for different scenarios. Example: embed a page showing an order detail into another page displaying a list of all orders. When you select an order from the list, it will be displayed in detail in the “inner” page.

The technology described in this part is very nice for modularising complex or large screens. It is not appropriate to use this technology for very fine modularisation, e.g. for just a couple of fields. (For more information on how to deal with “fine modularisation”, see the *Java Custom Controls* documentation.)

There are two controls which support embedding of pages into pages:

- The SUBCISPAGE2 control represents a rectangular area inside a page in which another Application Designer page can be included.
- The ROWTABSUBPAGES control is a tab selection control where a dynamic set of pages can be arranged.

The following topics are covered below:

SUBCISPAGE2 Control

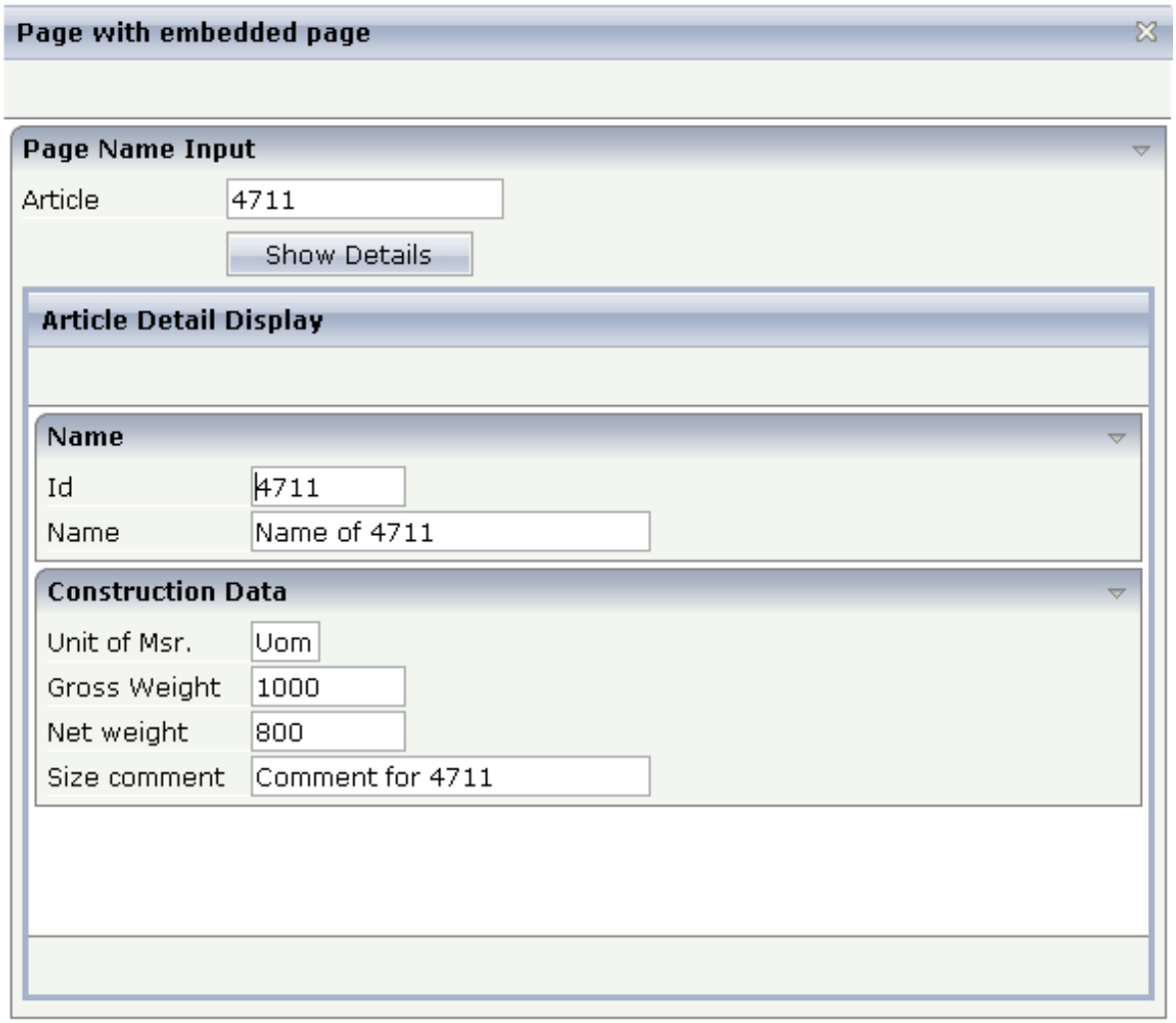
The SUBCISPAGE2 control allows you to place one page into another page. You may already have read the section describing the SUBPAGE control which allows to place any HTML page into an Application Designer page. The differences between the SUBCISPAGE2 and the SUBPAGE tag are:

- With SUBCISPAGE2, you embed Application Designer pages, not normal HTML pages.
- Application Designer pages are normally started using a servlet "StartCISPage" which creates an embedding frame in which the Application Designer page is placed. The SUBCISPAGE2 control automatically creates this frame, you do not have to take care of this.
- There is a defined communication channel allowing the “outside page” to interact with the “embedded page”, and vice versa.
- The embedded page is automatically linked to the Application Designer session management. It runs in the same session - and typically also in the same subsession as the embedding page.

- SUBCISPAGE2 Properties

Simple Example

The following example shows the input of an article number and its article detail data:



The detail data page is embedded into the whole (outer) page. The XML code of the outer page is:

```
<page model="OuterPageAdapter" pagename="Demo.html">
    <titlebar name="Page with embedded page">
    </titlebar>
    <header>
    </header>
    <pagebody takefullheight="true">
        <rowarea name="Page Name Input">
            <itr>
                <label name="Article" width="100">
```

```
                </label>
                <field valueprop="article" length="20">
                </field>
            </itr>
            <vdist height="5">
            </vdist>
            <itr>
                <hdist width="100">
                </hdist>
                <button name="Show Details" method="showDetails">
                </button>
            </itr>
            <vdist height="5">
            </vdist>
            <rowtable0>
                <itr width="100%">
                        <subcispage2 subcispageprop="innerPage" width="100%" ↩
height="350" borderwidth="1">
                        </subcispage2>
                </itr>
            </rowtable0>
            <vdist height="5">
            </vdist>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The SUBCISPAGE2 control references a property `innerPage` which is provided by the adapter class of the page. The height can be specified depending on the whole page's height or can be fixed.

The corresponding adapter source is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.SUBCISPAGEInfo;

public class SubCisPage2Adapter
 extends Adapter
{
    // property >innerPage<
    SUBCISPAGEInfo m_innerPage = new SUBCISPAGEInfo();
    public SUBCISPAGEInfo getInnerPage() { return m_innerPage; }

    // property >article<
    String m_article;
    public void setArticle(String value) { m_article = value; }
    public String getArticle() { return m_article; }
```

```
    /** */
    public void init()
    {
        m_innerPage.showPage("ArticlePage.html");
    }

    /** */
    public void showDetails()
    {
        // fetch adapter of inner page
        ArticlePageAdapter ipa = (ArticlePageAdapter) ↩
findAdapter(ArticlePageAdapter.class);
        ipa.init(m_article);

        // trigger a refresh of the innerpage
        m_innerPage.refreshContentOfCurrentPage();
    }
}
```

The property `innerPage` is of type `com.softwareag.cis.server.util.SUBCISPAGEInfo`. With method `SUBCISPAGEInfo.showPage`, the article page is started within the subarea. This does not have to be flexible all the time - but it may be on request. (Maybe there are several versions of displaying the detail data, depending on the article type).

When choosing the **Show Details** button, the method `showDetails()` is called. It prepares the adapter of the inner page to display the detail data of the requested article. Afterwards, the method `SUBCISPAGEInfo.refreshContentOfCurrentPage` is called in order to reload the embedded page. Consequently, the article details are shown.

See the JavaDoc documentation of class `SUBCISPAGEInfo`.

SUBCISPAGE2 Properties

Basic			
subcispageprop	Name of adapter property representing the control on server side. The property must be of type "TABSUBPAGESInfo". View the Java API Documentation for further information.	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").	Optional	100 120 140 160 180

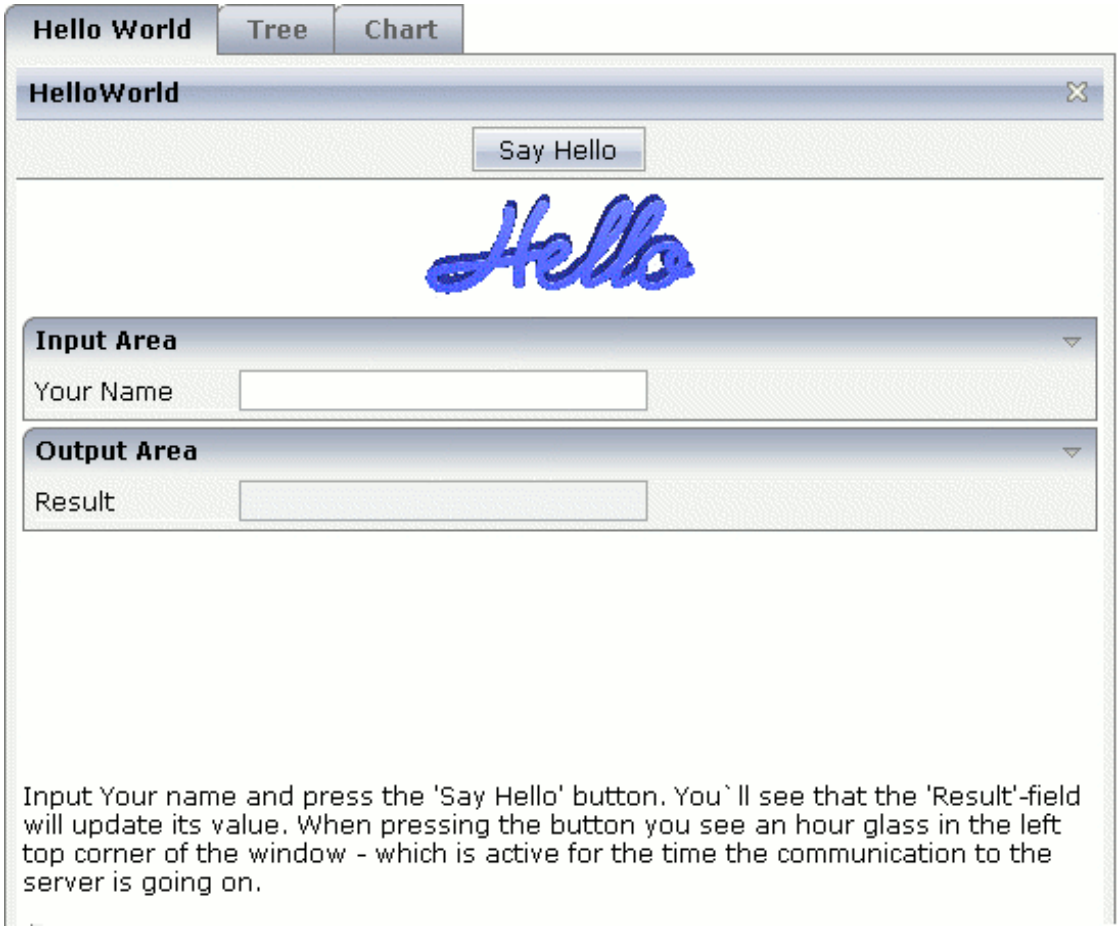
	(C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a width this control can reference. If you specify this control to have a width of 50% then the parent element (e.g. 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.		200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
height	(already explained above)		
borderwidth	Border size of control in pixels. Specify "0" not to render any border at all.	Optional	1 2 3 int-value
withownborder	Default is false. If WITHOWNBORDER is set to true, the subcispage2 control is rendered with its own 3D lookalike border. Set BORDERWIDTH to 0 if WITHOWNBORDER is set to true.	Optional	true false
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000	Optional	

	background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
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 synchronized 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
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 to span over more than one columns. The property only makes sense in table rows that are synchronized 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

ROWTABSUBPAGES Control

The ROWTABSUBPAGES control allows to switch between several Application Designer pages using tabs. The displayed number of tabs and names are derived dynamically from its adapter properties.

Optionally, the ROWTABSUBPAGES control may contain exactly one STRAIGHTTABPAGE as a subnode. STRAIGHTTABPAGE must be the first tab. This allows for combining ROWTABAREA behavior with ROWTABSUBPAGES behavior. Having a STRAIGHTTABPAGE as the first tab improves the loading behavior of ROWTABSUBPAGES. For an example, see the **80_straighttabpage** layout in the **cisdemos** project.



The XML definition is:

```
<pagebody>
    <rowtabsubpages pagesprop="tabpages" height="600" borderwidth="0">
    </rowtabsubpages>
</pagebody>
```

The ROWTABSUBPAGES control references a property `tabPages` which is provided by the adapter class of the page. The height can be specified depending on the whole page's height or can be fix. The page style can be manipulated directly.

The corresponding adapter source is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TABSUBPAGESInfo;

public class RowTabSubPageAdapter
    extends Adapter
{
```

```
    // property >tabpages<
    TABSUBPAGESInfo m_tabPages = new TABSUBPAGESInfo();
    public TABSUBPAGESInfo getTabpages() { return m_tabPages; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_tabPages.addItem("Hello World","HelloWorld.html");
        m_tabPages.addItem("Tree","trees_01.html");
        //m_tabPages.addItem("Dynamic Combo", "HelloWorld.html");
        m_tabPages.addItem("Chart", "HelloWorld.html");
    }
}
```

The property `tabPages` is of type `com.softwareag.cis.server.util.TABSUBPAGESInfo`. There are methods for adding and removing items from the `tabPages` object. See the JavaDoc documentation. The number of items can be changed at any time.

- Properties
- Performance Considerations

Properties

Basic			
pagesprop	Name of adapter property representing the control on server side. The property must be of type "TABSUBPAGESInfo". View the Java API Documentation for further information.	Obligatory	
triggerserver	Flag indicating whether the adapter should be triggered if the user switches between pages. If set to true, method trigger() inside the TABSUBPAGESInfo object is called - before switching the page. Therefore the adapter can abort a page switch - maybe a user has to enter some data first on the current page before switching to another one.	Optional	true false
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50%	Optional	100 150 200 250 300 250 400 50%

	then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		100%
scrollable	If set to "true" then small icons will appear on the right border of the control. If the size of the "tabs" is too big and some tabs are cut as consequence then you can use these icons for scrolling left and right.	Optional	true false
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".	Optional	
fastbufferswitch	If this property is switched to "true" (default is "false") then the contained subpages are buffered in a way that switching between tabs is not done by loading a new page but by just switching the visibility of pages. Please pay attention to that switching between pages in this case does not reload the page content from the server when switching! In order to enable fast switching you have to set the framebuffersize in cisconfig (n +1), n being the number of tabs to switch.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
borderwidth	Border width (in pixels) of the sub-page that is contained inside this control. Define "0" to avoid rendering any border.	Optional	1 2 3 int-value
leftindent	Inserts a horizontal distance left of the first "tab" and shifts the "tabs" to the right as consequence. The value you may define represents the number of pixels that are inserted.	Optional	1 2 3 int-value
paddingleft	Number of pixels which you want to keep as margin between the tab control's left border and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
paddingtop	Number of pixels which you want to keep as margin between the upper tab row and the inner sub page. Default is 5 pixel.	Optional	1 2

			3 int-value
paddingright	Number of pixels which you want to keep as margin between the tab control's right border and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
paddingbottom	Number of pixels which you want to keep as margin between the bottom of the tab control and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
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	

Performance Considerations

Many users like the subdivision of pages into “tabs”. Application Designer offers several controls for this - let us compare the ROWTABSUBPAGES control described in this section with the ROWTABAREA control described in the *Layout Elements* documentation.

The ROWTABAREA control has certain content areas (TABPAGEs) and always makes one of them visible. This means: the page has much more HTML code and controls then are visible. The size of the page is important for the performance of the page in the browser: the bigger the size, the longer it takes the browser to render a page (also if it is already cached).

The ROWTABSUBPAGES control offers a subpage in which you can place contained pages.

Now imagine that you have 500 fields to be displayed inside “tabs”: in this case, it is more performant to build one “mother page” containing the ROWTABSUBPAGES control and to have five “detail pages”, one for each “tab”, than having one big page with all 500 fields, arranged by a ROWTABAREA control.

In the demo workplace, there is an example in which you can “feel” the difference - please have a look!

Remark on Modularisation

This section describes one - important - technique for modularisation: embedding of pages into other pages.

This technique is useful for “rough granular” integration aspects: it is used to arrange pages with a dedicated task (e.g. the maintenance of an order) into other screens (e.g. an overview of all orders). Each page - both the “outer” page and the “inner” page - keep their “page behavior”, i.e. they are talking independently to the server.

The hour glass icon indicates that a page is talking to its server adapter. If an “outer” page refreshes its “inner” page, the “outer” page first talks to the server, and afterwards the “inner” page. Therefore, there is more than one roundtrip between the client and the server.

As a consequence, it does not make sense (and it is not intended at all by Application Designer) to build up fine granular integration scenarios in which a group of fields is defined as an embeddable unit being used in several screens. This is the job of controls to easily build up your own one. Controls which you build render a group of controls (e.g. an area for entering an address) and can be re-used in different pages. Controls always talk to the server within the same roundtrip. They are available as design time controls - if you change the behavior of one control definition, all pages using this control have to be regenerated.

It is comparable with C programming. You have libraries that you put directly into your compilation process. If the libraries change, you have to recompile. This is the level of controls. On the

other hand, you have units of rougher granularity: e.g. DLLs. These can be changed without letting your program know. This is the level of page integration.

94 Multi Frame Pages

Multi frame pages are a special set of pages. Normal pages represent a generated HTML page - a multi frame page represents a generated HTML frameset page.

What are Multi Frame Pages?

Multi frame pages are a special set of pages. Normal pages represent a generated HTML page - a multi frame page represents a generated HTML frameset page.

A multi frame page does not contain controls but frames in which other pages are positioned. Each frame is associated with an ID (called “target” in this section). A frame may be:

- a normal HTML page
- an intelligent Application Designer page
- a frameset itself containing frames

Multi frame pages are the preferred way of arranging Application Designer pages in a frameset. Besides enhanced possibilities of communication between frames, multi frame pages automatically take care of keeping all Application Designer frames inside the same session.

Definition of Multi Frame Pages

The definition of multi frame pages is done with the Layout Painter. When you create a new layout, a dialog appears in which you select a template. To create a multi frame page, you have to select the "Multi Frame Page" template. The Layout Painter will open just as usual, but instead of having the PAGE control as the highest control, you now see the control MFPAGE. You can reach a number of controls that are related to multi frame page management.

The following controls are “normal frame controls” (they are described below):

- MFPAGE - the top element of multi frame pages.
- MFCISFRAME - a frame in which an Application Designer HTML page is loaded.
- MFHTMLFRAME - a frame in which a normal HTML page is loaded.
- MFFRAMESET - an area that can be subdivided into frames itself.

The following controls are “workplace controls”. The Application Designer workplace is based on these controls.

- MFWPFUNCTIONS
- MFWPACTIVEFUNCTIONS
- MFWPCONTENT

MFPAGE

The MFPAGE is the top node of every multi frame page. It can be subdivided into frames or framesets.

Basic			
separation	Specifies how the corresponding internally used frameset is subdivided: choose "rows" for subdividing into rows, "cols" for subdividing into columns.	Obligatory	rows cols
sizing	Defines the size of the contained sub-frames. If you have three sub-frames to show up inside the page then you might specify "200,200,*" to specify how the height (if SEPARATION is "rows") or the width (if SEPARATION is "cols") is distributed among the frames. You can specify per frame either a pixel value or a "*".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
border	Space between frames contained in the frameset that is internally built up.	Optional	1 2 3 int-value
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
frameborder	Defines if to display a border around the contained frames. Valid values are "true" or "false".	Optional	true false

framespacing	Defines the amount of additional space between the frames. Value is a pixel value.	Optional	1 2 3 int-value
framesetstyle	Style passed to the HTML-frameset definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

MFCISFRAME

The MFCISFRAME represents a frame in which an Application Designer page is shown. The name of the page is passed as a parameter.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specify with the TARGET property.	Obligatory	
cisurl	URL of the page to be shown inside. Use /project/page.html as syntax, e.g. "/HTMLBasedGUI/empty.html". Do NOT use only page.html believing that you do not have to specify the project because the multi frame page runs in the same project than the page you want to open - you ALWAYS have to specify the project!	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence.	Optional	true false

	Valid values are "true" and "false". Default is "true".		
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
withownborder	Flag that indicates if started pages show an own border. Default is false.	Optional	true false
Unload Behaviour			
unloadbehaviour	Reaction that CIS should take if the page inside the frame is closed. Possible values are "NOTHING" for doing nothing and "REMOVESESSION" for removing the session on server side.	Optional	NOTHING REMOVESESSION

	Do not define this property just "by accident" but leave it to the default ("NOTHING"). You only switch to "REMOVESESSION" if you want that the server side session is destroyed when leaving the page. This is the case if you have one page that clearly indicates the closing of a session at the point of time when the page is closed.		

Applications can change the page that is shown inside the MFCISFRAME by using the method `Adapter.openCISPageInTarget(...)`.

MFHTMLFRAME

The MFHTMLFRAME represents a frame in which a normal HTML page is shown. This page can be a static HTML page or any URL - e.g. a URL referring to a certain JSP page.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specify with the TARGET property.	Obligatory	
url	URL to be opened inside the frame. The URL can be defined relative to the multi frame page or can be defined in an absolute way.. Example: You can define "../HTMLBasedGUI/workplace/header2.html" - or "http://www.softwareag.com".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence. Valid values are "true" and "false". Default is "true".	Optional	true false
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false

scrolling	Boolean that indicates whether the frame can be scrolled. Default is true.	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value

MFFRAMESET

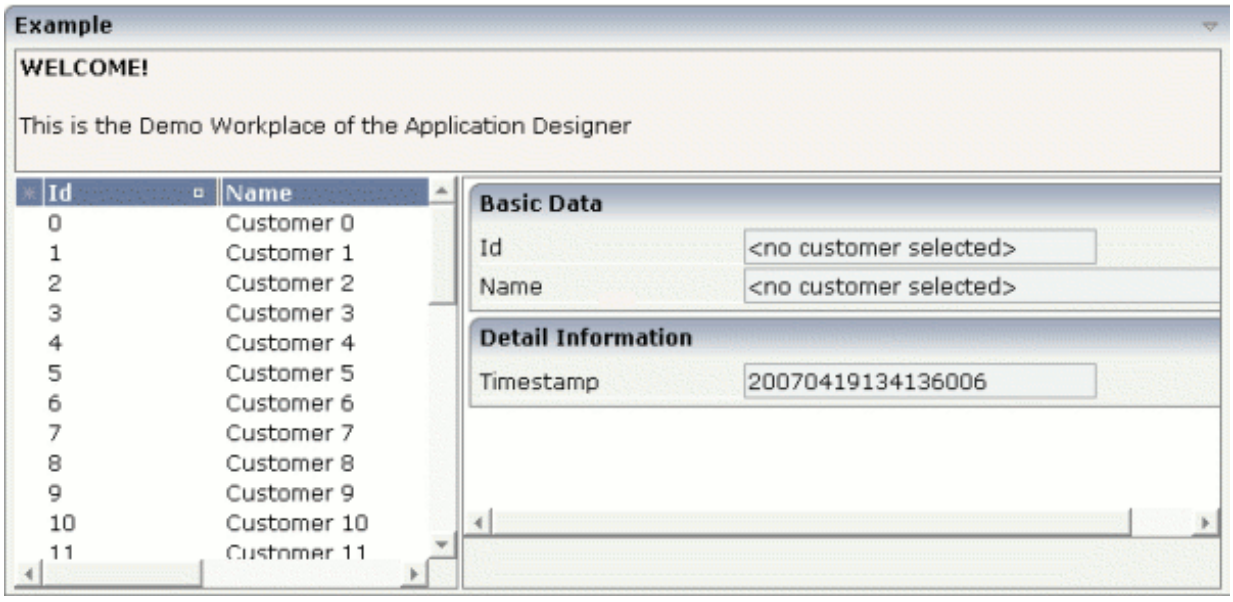
The MFFRAMESET represents a frame that is internally again divided into frames. The MF-FRAMESET definition decides whether to divide into rows or columns, and how to size the inner frames.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specify with the TARGET property.	Obligatory	
separation	Specifies how the corresponding internally used frameset is subdivided: choose "rows" for subdividing into rows, "cols" for subdividing into columns.	Obligatory	rows cols
sizing	Defines the size of the contained sub-frames. If you have three sub-frames to show up inside the page then you might specify "200,200,*" to specify how the height (if SEPARATION is "rows") or the width (if SEPARATION is "cols") is distributed among the frames. You can specify per frame either a pixel value or a "*".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
border	Space between frames contained in the frameset that is internally built up.	Optional	1 2 3 int-value
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
frameborder	Defines if to display a border around the contained frames. Valid values are "true" or "false".	Optional	true false
framespacing	Defines the amount of additional space between the frames. Value is a pixel value.	Optional	1

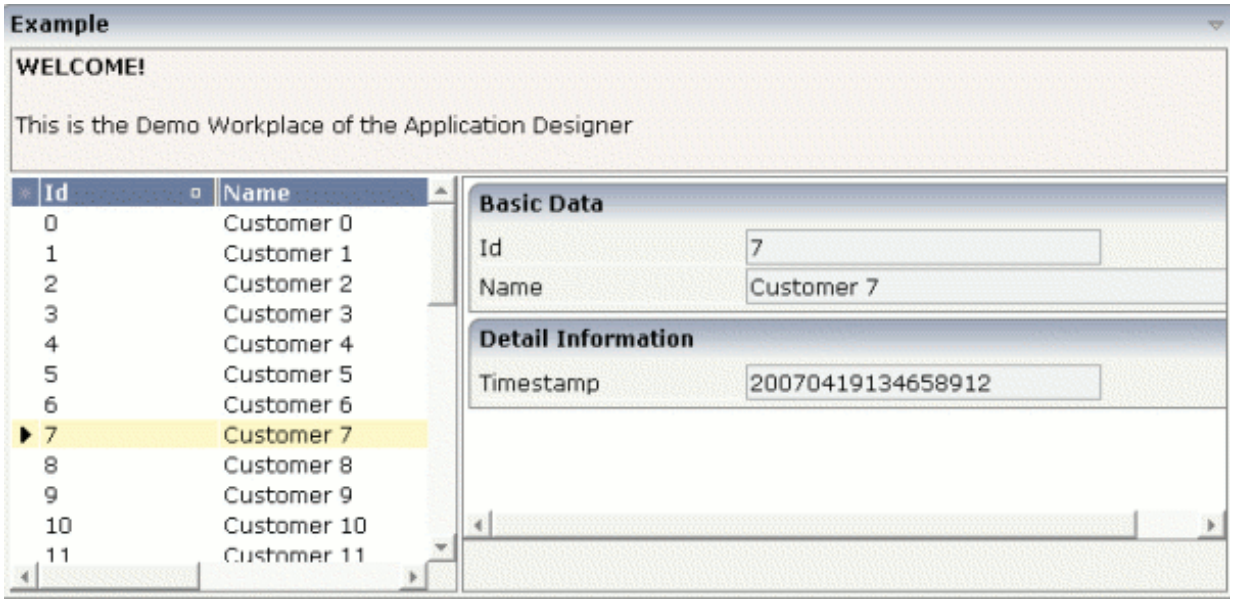
			2 3 int-value
framesetstyle	Style passed to the HTML-frameset definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

Example

The example that will be built in this section produces the following output:



When selecting a customer on the left, the customer detail screen is displayed on the right:



When the user selects another record on the left, the screen on the right is updated accordingly.

- The Multi Frame Page Around
- The Left Frame
- The Right Frame

The Multi Frame Page Around

First let us have a look at the multi frame page itself. The layout definition is as follows:

```
<mfpage separation="rows" sizing="70,*">
    <mfhtmlframe target="HEADER"
                 url="../HTMLBasedGUI/workplace/welcome.html"
                 resizable="true"
                 withborder="false"
                 scrolling="false"
                 framestyle="border: 1px #808080 solid">
    </mfhtmlframe>
    <mfframeset target="AROUND"
                separation="cols"
                sizing="200,*">
        <mfcisframe target="INNERLEFT"
                    cisurl="/cisdemos/25_mfinnerleft.html"
                    framestyle="border-right: 1px solid #808080;
                    border-bottom: 1px solid #808080">
        </mfcisframe>
        <mfcisframe target="INNERRIGHT"
                     cisurl="/HTMLBasedGUI/empty.html"
                     framestyle="border: 1px solid #808080">
        </mfcisframe>
```

```
    </mfframeset>
</mfpage
```

The page is subdivided into three frames: "HEADER", "INNERLEFT" and "INNERRIGHT". Two of them are Application Designer frames, one is an HTML frame. Every frame is pointing to a certain page.

The Left Frame

The INNERLEFT frame's page displays a text grid and lets the user select from the list of items. The layout definition is:

```
<page model="MFInnerLeftAdapter">
    <pagebody horizdist="false" takefullheight="true">
        <itr height="100%" fixlayout="true" width="100%">
            <textgrid2 griddataprop="customers" width="100%" height="100%" ↩
selectprop="selected"
                       singleselect="true" hscroll="true" ↩
directselectmethod="onSelect"
                       directselectevent="onclick">
                <column name="Id" property="id" width="100">
                </column>
                <column name="Name" property="name" width="400">
                </column>
            </textgrid2>
        </itr>
    </pagebody>
</page>
```

The adapter implementation is done in the following way:

```
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.ServerLog;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

public class MFInnerLeftAdapter
 extends Adapter
{
 // ------------------------------------------------------------------------
 // inner classes
 // ------------------------------------------------------------------------

 public class CustomerInfo
 {
            boolean m_selected;
            String m_id;
            String m_name;
            public String getId() { return m_id; }
            public String getName() { return m_name; }
            public boolean getSelected() { return m_selected; }
```

```
            public void setId(String string) { m_id = string; }
            public void setName(String string) { m_name = string; }
            public void setSelected(boolean b) { m_selected = b; }
}
// ------------------------------------------------------------------------
// members
// ------------------------------------------------------------------------

TEXTGRIDCollection m_customers = new TEXTGRIDCollection();

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

public TEXTGRIDCollection getCustomers() { return m_customers; }

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

public void init()
{
 super.init();
 for (int i=0; i<40; i++)
 {
  CustomerInfo info = new CustomerInfo();
  ci.setId(""+i);
  ci.setName("Customer " + i);
  m_customers.add(ci);
 }
}

public void onSelect()
{
 try
 {
  CustomerInfo info = (CustomerInfo)m_customers.findLastSelectedItem();
  // prepare adapter of right frame
  MFInnerRightAdapter mfira =
                    (MFInnerRightAdapter)findAdapter(MFInnerRightAdapter.class);

  mfira.prepare(ci.getId());
  // preload adapter so that only one request is executed
  includeAdapterInResponse("../_DevelopersGuide/mfinnerright.html",false);
  // refersh target
  refreshTarget("INNERRIGHT");
 }
 catch (Throwable t) { ServerLog.appendException(t); }
}
}
```

The class contains the following:

- An inner class for the text grid items.
- An `init` method for filling the text grid.
- A `onSelect()` method that is called when the user selects a text grid line.

The "critical" lines of code are inside the `onSelect()` method. Inside the method

- the selected line is determined,
- the adapter of the right neighbor screen is prepared so that it shows the data of the selected line,
- the right page is switched to the detail page (if first call) or
- the right page is refreshed to present the correct adapter information.

The Right Frame

The right frame is loaded with */HTMLBasedGUI/empty.html* first. With the first selection in the text grid, the detail page is opened inside the right frame. Afterwards, the detail page is refreshed to update its content.

Communication between Frames

You already saw some methods in the **previous** section enabling one frame to open pages in another frame and to refresh information of other frames.

API inside the Adapter Class

The following table shows a summary of functions that you can reach in your adapter class which inherits from `com.softwareag.cis.server.Adapter`. See the JavaDoc documentation for implementation details.

Method	Description
`openCISPageInTarget(...)`	Opens a certain Application Designer HTML page inside a certain frame which is identified by its target ID. There is a set of methods with different parameter notation. The default method just needs to know the page URL and the ID of the frame. Other methods expect more information, e.g. if you want to open the Application Designer page in a different subsession.

Method	Description
`refreshTarget(...)`	Refreshes the target's frame content. This method is to be used if you want the target frame not to change its page but to update its content. In the **example** in the previous section, this method is used after having updated the right frame's adapter on the server side.
`invokeMethodInTarget(...)`	Invokes a method in the target frame's Application Designer HTML page. The call is triggered from the client - for example, imagine that a button supporting the method is pressed in the target frame's Application Designer HTML page.
`sizeTarget(...)`	Manipulate the size of the target. Each target gets a certain size by the frame set definition on top of it (e.g. if the frame set definition has a sizing of "200,300,*", then the second frame has a size of "300". You can change the size of a target by using this method.

Pay Attention to Request Processing

Be aware of the request processing in the browser: only the page which sends a request (e.g. the left page in the **example**) is the active page and will process the response. All other pages living in neighboring frames are by default not affected.

Consequence: if you want to change or refresh these pages, you have to explicitly do so using the API presented in one of the previous sections.

Important: The adapter that processes the request is the one to call the API methods.

Session Management (I)

Maybe you have already tried to build multi frame pages on your own, using HTML framesets:

```
...
...
<frameset cols="*,*">
  <frame src="../servlet/StartCISPage?PAGEURL=/project/left.html">
  <frame src="../servlet/StartCISPage?PAGEURL=/project/right.html">
</frameset>
...
...
```

If so, you will have seen that in each of the frames, the Application Designer page will be opened correctly. However, both pages are running in independent sessions (not subsessions).

Opening the same pages using Application Designer's MF* controls (MFFRAMESET, MFCISPAGE) will keep both pages inside the same session and subsession.

Note: Details on session management are provided in the section *Session Management*.

Session Management (II)

When communicating between frames, e.g. by using the method `Adapter.openCISPageInTarget()`, the default is that the page that is opened in another target will be opened in the same session/subsession as the one that initiated the frame communication. Session ID and subsession ID are taken over by default.

There are certain variants of the `openCISPageInTarget()` method that allow to control the management of a subsession in a more fine granular way: you may pass as parameter the ID of the subsession in which a page should be opened in another page; i.e. you can explicitly decouple the other frame's subsession from your own one.

The workplace that comes with Application Designer makes use of this: every time you open a content window, this content is managed in its own subsession, being decoupled from the workplace's subsessions and being decoupled from other content pages' subsessions.

Use these functions with care: typically all application adapters should run in one subsession, and only an "outside function" (such as the workplace management) should take care of starting various contents in various subsessions.

Combination with Normal Application Designer Pages

There is no problem to integrate multi frame pages into other Application Designer pages. The mechanisms described in the section ***Embedding Pages into Pages*** are valid for both normal Application Designer pages and multi frame Application Designer pages.

This means:

- You can embed multi frame pages into normal Application Designer pages via the SUBCISPAGE2 control.
- You can embed multi frame pages into normal Application Designer pages via the ROWTABSUBPAGES control.

95 Embedding Pages into a Workplace

In the *First Steps*, you learned already how to build pages that are generated by the Layout Painter. This part explains how to integrate Application Designer pages into workplace/portal environments. There are different scenarios that are described here:

- Usage of the Application Designer workplace framework that lets you design and implement individual workplaces for your application.
- Integrating Application Designer pages into various portal scenarios by opening them with a URL.
- Writing a workplace framework on your own - i.e. use Application Designer in order to build your workplace, but not on base of the Application Designer workplace framework.

The following topics are covered below:

Integration into Other Workplace/Portal Scenarios

In many cases, you want to run Application Designer pages inside your own environments that are outside of Application Designer.

The requirements of other workplace/portal environments are:

- Pages must be accessible by URLs.
- There must be a possibility to pass information to pages. For example, user management is provided by a portal management. The result (the name of the user who is currently logged in) should be passed to applications for further processing.

To call Application Designer pages with a URL: normally each single Application Designer page can be called individually with a corresponding URL. “Normally” means that this is true from the Application Designer perspective - maybe it is not completely true from your application’s perspective: one page requires a certain page to be run first, etc.

To call an Application Designer page, simply use the following URL:

```
http://<host>:<port>/cis/StartCISPage?PAGEURL=<pageURL>
```

Replace the `<pageURL>` with the URL of the wanted Application Designer page and it will be opened.

For information on additional parameters that you can pass via the StartCISPage servlet, see *Appendix E - StartCISPage Servlet* .

Passing Parameters to your Application Designer Page

You can append any number of parameters to the URL mentioned in the previous section. Each parameter consists of the sequence "&<*paramName*>=<*paramValue*>". If you want to pass the `customerId` to a “customer detail” page, the URL would look like:

```
http://<host>:<port>/cis/StartCISPage?PAGEURL=/appxyz/customerdetail.html&customerId=4711
```

Each parameter is bound to a corresponding property of the page adapter. For example, the “customer detail” page is hooked on the adapter `CustomerDetailAdapter`. Therefore, it must provide a corresponding `customerId` property to which the parameter is passed at runtime:

```
public class CustomerDetailAdapter extends Adapter
{
    ...
    public void setCustomerId(String value)
    {
        ...
    }
    ...
}
```

Extended Functions in the Application Designer Workplace

The **previous** section covered the “normal usage mode” of the Application Designer workplace. But there are extended functions allowing you to more interactively operate with the Application Designer workplace.

These functions include:

- Drag-and-drop interface: you can drag and drop icons within the hierarchy of the workplace. You can drop information that was dragged from any content page into the workplace.
- Right mouse button interface on workplace nodes.

These functions allow the users to arrange their workplace settings (e.g. functions that are part of their workplace) in a simple way on their own.

- Interface IMFWorkplaceEventListener

- Example

Interface IMFWorkplaceEventListener

The base of the extended functions is the interface `com.softwareag.cis.workplace.IMFWorkplaceEventListener`. The interface contains methods that are called on certain events. The most important methods are:

- `reactOnContextMenuRequest`
  This method is called when a user presses the right mouse button on a tree node of the workplace. Your implementation can build up a context menu - just as normal context menus are built up inside the tree management.
- `reactOnDrop`
  This method is called when the user performs a drag-and-drop operation inside the workplace. Your implementation may copy the dragged nodes below the dropped node or may open a pop-up menu in which the user is asked about what to do with the dragged items.
- `reactOnDropGeneric`
  This method is called when the user performs a drag-and-drop operation from any DROPICON control that is part of content pages.

The implementation of the interface is completely “yours”. Use the workplace interface you got to know in the previous section to manipulate the workplace, e.g. to access the currently shown tree and to manipulate it.

An instance of the workplace event handler is passed by calling the method `registerMFWorkplaceEventListener` inside the `MFWorkplaceInfo` class:

```
MFWorkplaceInfo workplaceInfo = new MFWorkplaceInfo("/HTMLBasedGUI/empty.html",
                                    "../softwareag/styles/CIS_DEFAULT.css"); ↩

workplaceInfo.setSynchTabNavigation(true);
workplaceInfo.registerMFWorkplaceEventListener(new XYZ(...));        ↩
```

See the Java API documentation for detailed information.

Example

Among other features, the Application Designer demo workplace framework provides the following:

- Right mouse button click on a workplace menu item (copy, cut, paste, etc.).
- Drag-and-drop within the workplace menu (to move menu items).

Have a look at the event listener source coding. You can find it in your installation at:

<installdir>/cis/cisdemos/src/com/softwareag/cis/demoworkplace/CISDemoWorkplaceEventListener

Building Own Workplaces as a Frameset Definition

A set of functions is available which simplify the usage of Application Designer HTML pages inside a given HTML frameset definition. The functions are not only usable in the scope of work-place/portal management, but can also be used apart from this.

Basics

The basic functions cover the following aspects:

- You can define an HTML page containing any kind of frameset you want. In this page, you design the frames, their sizes, their scroll behavior, their behavior when resizing the screen, etc. For each frame with which you want to interact, you define an identifier name.
- You open Application Designer pages inside the frames. There are two possibilities:
  1. Open these pages with a URL as described in the previous section.
  2. Open these pages with adapter methods (server-side processing).

This section will focus on the second possibility since the first is just a certain usage of what is described in the previous section. This offers you an explicit control about what happens inside the frames: e.g. a page within frame "A" should be replaced by another page. Before proceeding, the user should be asked whether to store unsaved data (or not).

It is possible to communicate with frames on the client side. This means, you can build up interaction (e.g. you want to update another frame's content) without any flickering in the target frame.

Defining the Frameset

In the following screen, a page is shown which is divided into three frames:

The corresponding frameset definition of the page is:

```
<html>

<head>
<title>New Page 2</title>
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
</head>

  <frameset cols="200,*">
    <frameset rows="*,*">
      <frame name="lefttop" ↩
src="/cis/servlet/StartCISPage?PAGEURL=/cisdemos/frameleft.html">
      <frame name="leftbottom" src="blank.html">
    </frameset>
    <frame name="right" src="blank.html">
  </frameset>

</html>
```

The frameset contains three frames with the IDs `lefttop`, `leftbottom` and `right`. The `lefttop` frame opens the Application Designer page */cisdemos/frameleft.html*. This page contains buttons for some functions and acts like a “menu page”.

Simple Way of Opening Pages in Frames

When choosing the **Customer Orders** button, the corresponding Application Designer page is opened in the `leftbottom` frame:

The page shows a list of customer orders. It is a normal Application Designer page. How can it be opened by choosing the **Customer Orders** button?

The */cisdemos/frameleft.html* page (acting as a "menu page") is hooked on to a Java adapter class which looks as follows:

```
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class FrameLeftAdapter
    extends Adapter
{
    /** */
    public void onArticleMaster()
    {
        // TODO Auto-generated method stub
    }

    /** */
    public void onCustomerMaster()
    {
        // TODO Auto-generated method stub
    }
```

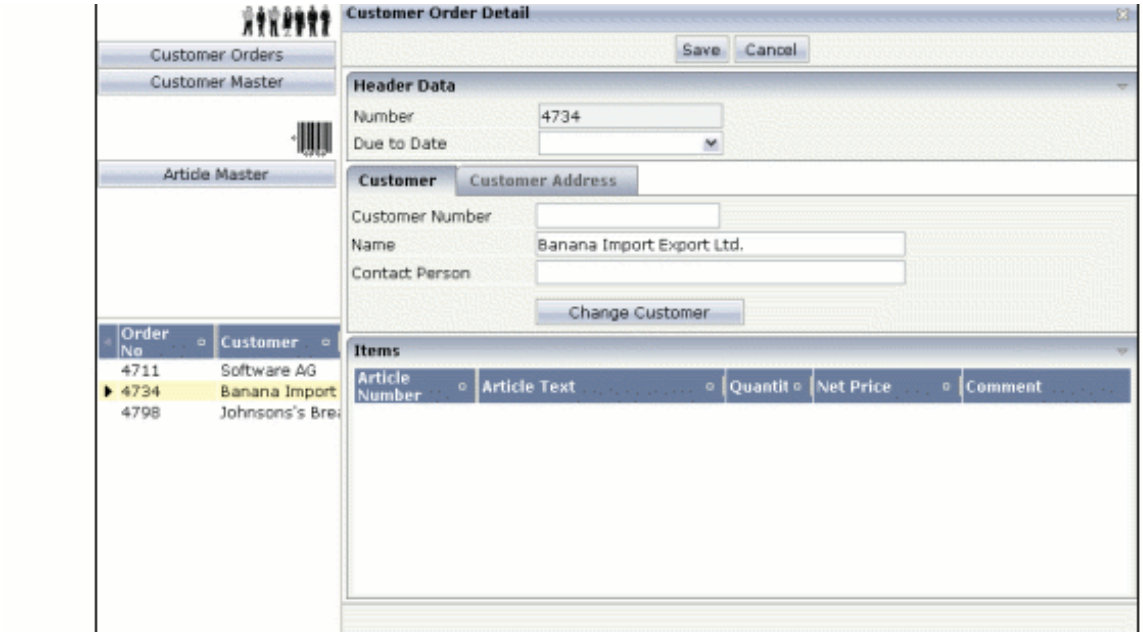
```
    /** */
    public void onCustomerOrders()
    {
        this.openCISPageInTarget("OpenCustomerOrders.html", "leftbottom");
    }
}
```

By choosing the **Customer Orders** button, the method `onCustomerOrders` is called. This method performs a method `openCISPageInTarget` inherited from class `Adapter`. The first parameter of the method is the page that is to be opened; the second parameter defines the ID of the frame in which the page is to be opened.

The page *OpenCustomerOrders.html*, which is opened when choosing the **Customer Orders** button, is running inside the same subsession as the page from which it was called. If you need to access the page adapter before opening the page inside the "leftbottom" frame, use the `findAdapter` method inside your adapter.

A More Complex Way of Opening Pages in Frames

When selecting an order in the `leftbottom` area of the previous example, a customer order page is displayed in the right frame:



The data from the order you selected is transferred into the corresponding fields of the customer order page. Have a closer look at the details.

This is the source of the adapter for listing customer orders:

```
import java.util.Iterator;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IInteractionSessionMgr;
import com.softwareag.cis.server.InteractionSessionMgrFactory;
import com.softwareag.cis.server.util.SelectableLine;
import com.softwareag.cis.server.util.TEXTGRIDCollection;
import com.softwareag.cis.util.CDate;

public class OpenCustomerOrdersAdapter
    extends Adapter
{
    // ------------------------------------------------------------------------
    // inner classes
    // ------------------------------------------------------------------------

    public class Order
        extends SelectableLine
    {
        public Order(String number, String date, String customer)
        {
            m_customer = customer;
            m_date = new CDate(date);
            m_number = number;
        }

        // property >orders[*].customer<
        String m_customer;
        public String getCustomer() { return m_customer; }
        public void setCustomer(String value) { m_customer = value; }

        // property >orders[*].date<
        CDate m_date;
        public CDate getDate() { return m_date; }
        public void setDate(CDate value) { m_date = value; }

        // property >orders[*].number<
        String m_number;
        public String getNumber() { return m_number; }
        public void setNumber(String value) { m_number = value; }
    }

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

    // property >orders<
    TEXTGRIDCollection m_orders = new TEXTGRIDCollection();
    public TEXTGRIDCollection getOrders() { return m_orders; }

    // ------------------------------------------------------------------------
    // public adapter methods
```

```
    // ---------------------------------------------------------------------

    public void onOrderSelect()
    {
        // find the selected item
        Order selectedOrder = null;
        Iterator iter = m_orders.iterator();
        while (iter.hasNext())
        {
            selectedOrder = (Order)iter.next();
            if (selectedOrder.getSelected() == true)
                break;
            else
                selectedOrder = null;
        }
        if (selectedOrder == null)
            return;
        // session management: "refresh" subsession
        String sessionId = this.m_interactionProcess.getSessionId();
        IInteractionSessionMgr iism = ↩
InteractionSessionMgrFactory.getInteractionSessionMgr();
        iism.removeSubsession(sessionId,"subsession_right");
        iism.createNewSubsession(sessionId,"subsession_right");
        // prefetch and manipulate adapter inside the refreshed subsession
        CustomerOrderDetailAdapter coda = ↩
(CustomerOrderDetailAdapter)iism.findAdapterInSubsession
            (sessionId, // sessionID
             "subsession_right", // subsessionId
             CustomerOrderDetailAdapter.class.getName(), // class
             "", // pageId, typically ""
             findPageApplication()); // application project
        coda.setNumber(selectedOrder.getNumber());
        coda.setName(selectedOrder.getCustomer());
        // navigate to page
       openCISPageInTarget("CustomerOrderDetail.html","subsession_right","right");
    }

    // ---------------------------------------------------------------------
    // standard adapter methods
    // ---------------------------------------------------------------------

    // property >messageType< implemented in Adapter
    // property >messageShortText< implemented in Adapter
    // property >messageLongText< implemented in Adapter

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_orders.add(new Order("4711","20020706","Software AG"));
        m_orders.add(new Order("4734","20020702","Banana Import Export Ltd."));
        m_orders.add(new Order("4798","20020604","Johnsons's Bread"));
```

```
    }
}
```

With method `onOrderSelect`, the selected line is determined first.

In the next steps, frame communication is prepared and finally done. The difference to the previous "simple" scenario is that the page which is opened runs in a different subsession inside the session management of Application Designer.

Remember that each browser instance internally requests one session, and that each session is divided into various subsessions. Adapters are running inside subsessions. The subsession is responsible for keeping and releasing resources. It corresponds to one interaction process which has a defined life cycle - e.g. the data input of a customer order. For more information, see the section ***Session Management***. Each subsession has an identifier - in this example, the name of the subsession is `subsession_right`. You can also create a unique ID with the class `com.softwareag.cis.util.UniqueIdMgmt`.

Our example program first removes the subsession `subsession_right`. Everything which is currently managed inside the subsession will be released. Since there is no subsession when being called the first time, no error will occur.

After releasing this subsession, a new subsession is immediately created. With the interaction session manager, you can access a method which passes back an adapter instance inside a given subsession. Like the method `findAdapter` of class `Adapter`, this method returns an adapter object which is managed inside the same subsession in which the adapter is running. With the interaction session manager, you can also access adapters inside different subsessions.

The returned adapter instance gets the selected data. Finally, the frame communication takes place: pay attention that the ID of the subsession has to be passed inside the `openCISPageInTarget` method.

When to Use the Complex Way

The complex way should be your "standard thinking" in this scenario. When dealing with Application Designer pages inside different frames, you have to take care about how you manage your sessions at the server side.

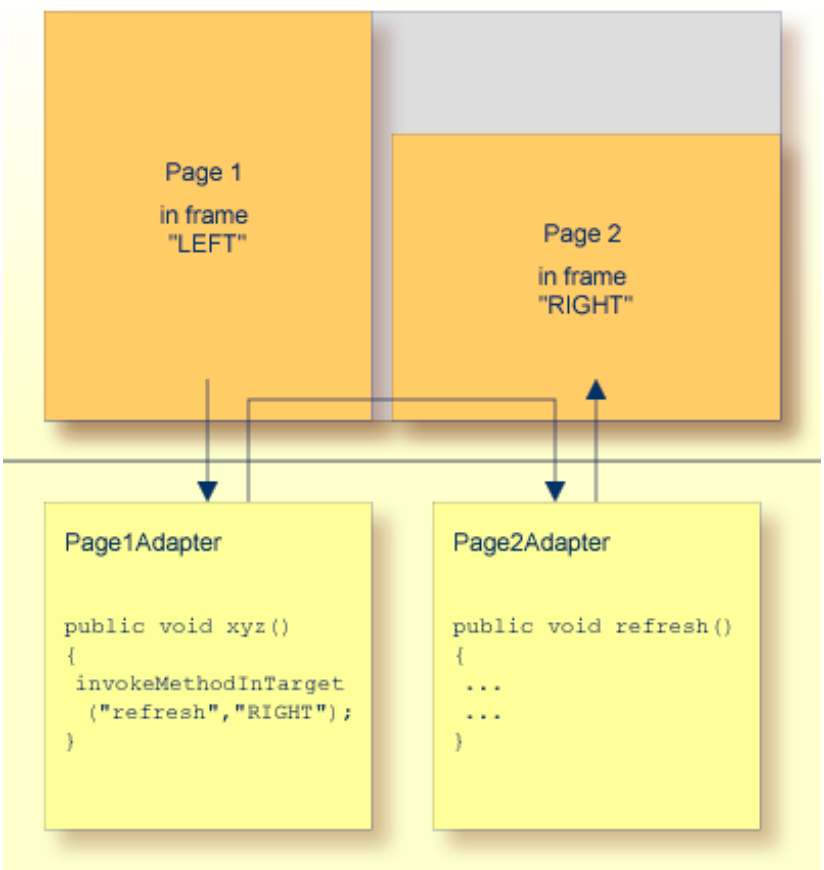
The content which runs inside the frames (e.g. the Customer Order screen) is not aware of these session management dependencies. But the designer of the workplace has to take care of the interaction possibilities inside the workplace.

Opening Normal HTML Pages inside Frames

In addition to the methods `openCISPageInTarget`, there is an equivalent method `openPageInTarget` which you inherit from the `Adapter` class. This page opens a normal HTML page inside one frame.

Frame Communication

When working with frames, it is possible to open a client-side communication channel between frames: by the client, you call an adapter method of an Application Designer page which is opened in a different frame. This is done by the method `invokeMethodInTarget(methodName,targetName)` which you inherit from the `com.softwareag.cis.server.Adapter` class.



This sounds strange at first: in the adapter processing of frame `LEFT`, you call an adapter method of frame `RIGHT` and the call is executed by the client - what is the reason for this? The advantage of calling the method by the client is that the call is initiated by the page which runs inside the target frame. This means that the target frame sends the request for method execution and updates its content as a reaction of the request's response. In other words: you can update pages in other frames without redrawing the page, i.e. without flickering.

This is a very powerful way of allowing communication between frames - but there are some restrictions which you have to keep in mind:

- The frame which initiates the interaction as well as the target frame must be in the same frameset page - in other words: they must share the same “document parent”.
- The page shown in the target frame must be received by the same server and port as the page which initiates the communication. Otherwise, you will receive a JavaScript security exception - it is (by default) not allowed to establish a client communication between pages coming from different hosts.

The frame communication framework acts upon error situations in a quite tolerant way:

- If you invoke a method inside a frame target and the frame does not exist, nothing will be done on the client side.
- If you invoke a method inside a frame target and there is no valid Application Designer page inside the frame, nothing will be done.
- If you invoke a method inside a frame target and the corresponding “target adapter” does not provide the requested method, the content of the frame target’s page will be synchronized with its adapter. This is similar to defining a BUTTON control and specifying a method which does not exist inside the adapter.

Tip: Only use frame communication if you want to update the content of another frame page. Do not use this method for “normal” interaction between adapters, without any changes on the page.

Multiple Frame Operations

You can call the methods `openCISPageInTarget`, `openPageInTarget` and `invokeMethodInTarget` multiple times inside one request, for example, if there are multiple frames you want to manipulate at the same time.

When Building your Own Workplaces

As you learned in this section, Application Designer provides powerful mechanisms to build flexible workplace/portal scenarios. Be aware that workplace management means more than bringing up some pages into different frames. Workplace management also means, for example, that you take care of opening and closing the session state (in Application Designer: subsession state) and that you have to provide global data (like the currently logged in user) shared by all adapters, etc.

X Working with PDF Documents

Creating PDF is a standard issue when providing user interfaces through the browser. This is the reason why Application Designer introduced a couple of PDF services. The Application Designer ideas which are applied in the area of HTML-based GUIs for applications are transferred into the generation of PDF output:

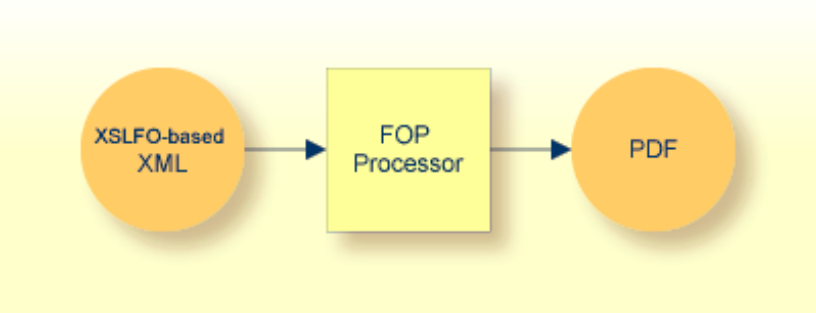
- High-quality output.
- Simple usage but still making control possible in a fine granular way on low level.
- Simple binding of PDF output to application data objects (adapters).
- XML as base for defining templates.

Introduction	General information on Application Designer's FOP (Formatting Objects Processor) services.
The First PDF Document	Explains the steps for creating a PDF document that is rendered inside the browser.
Printing	Explains how to produce paper printouts.

96 Introduction

FOP - Formatting Objects Project

There is a series of accepted standard modules coming from the Apache Group that form the base of Application Designer's PDF services. The so-called FOP standard (Formatting Objects Processor) is - though still in evolution - a healthy base for generating PDF documents through a certain XML-based specification: XSLFO. XSLFO consists out of a number of XML tag definitions by which a document is described. The XML definition is processed by a transform processor that produces PDF as output.

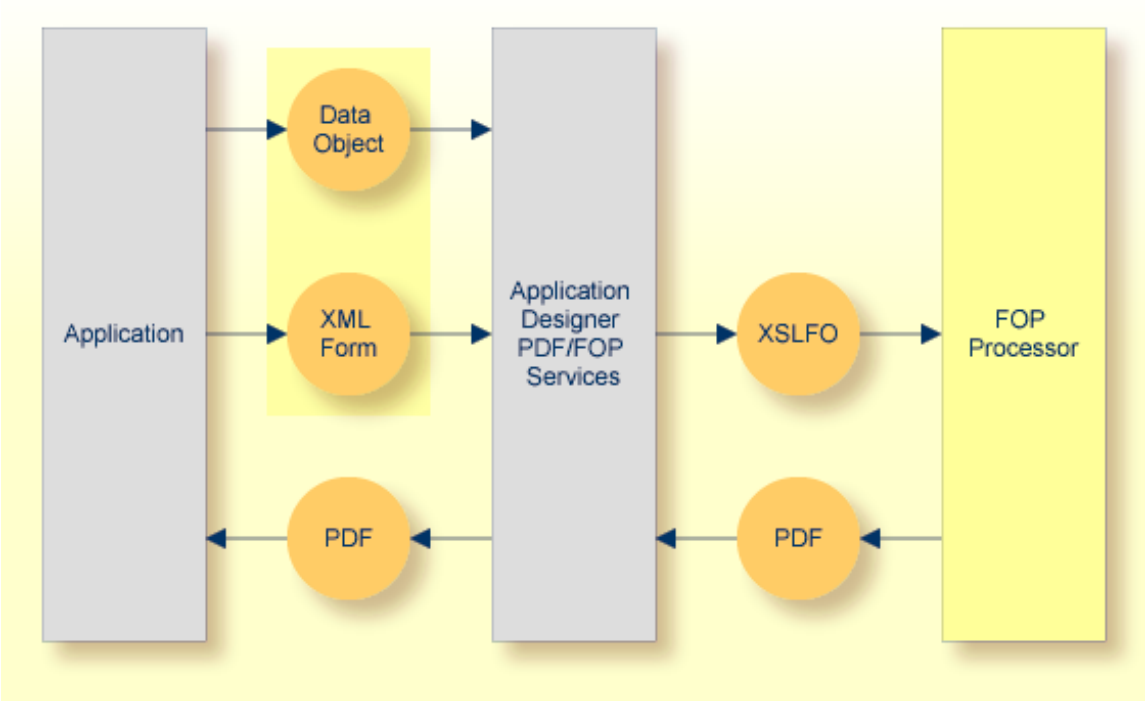


Though clear in concept and flexible in usage, XSLFO has a certain complexity: the number of tags and the number of properties inside the tags is quite high. Therefore, the effort for writing XSLFO-based XML documents is high and requires a profound understanding of FOP.

In addition, XSLFO is a pure output protocol - it describes a document before being transferred to PDF. The task of building the document and filling the right data in is not covered by FOP. This is where Application Designer's services come in.

Application Designer Services in Front of FOP

The following image summarizes the architecture of Application Designer's FOP services:



An application passes an XML form and a data object to the PDF/FOP services.

The XML form contains:

- Special abbreviation tags provided by Application Designer that simplify the creation of standard documents.
- Special tags provided by Application Designer that represent certain output controls (such as a grid) and contain binding information to a data object.
- Any XSLFO tag that you require. This means, you can directly use XSLFO statements and embed them into the form definition.

The data object (adapter) contains:

- All the “net data” that is mixed into the form. Net data is provided in the same way as you may be used to from working with Application Designer HTML pages: by corresponding properties that are referenced from the XML form definition.

The Application Designer PDF/FOP services build proper XSLFO out of the form and the data object. The XSLFO is passed to the normal FOP processing. The result is a PDF stream that is passed back to the calling application.

Output Formats

FOP is an abstract document definition. It is not bound to PDF. Application Designer offers both interfaces to create FOP-XSLO documents by the use of forms and it offers interfaces for printing FOP to a printer.

You can use other FOP output formats as well. Application Designer offers to you to directly generate the XSLFO result and pass it back to you as string.

Printing of Application Forms

Printing of application forms can be done in two ways:

- Server side PDF creation.
- Server side printing.

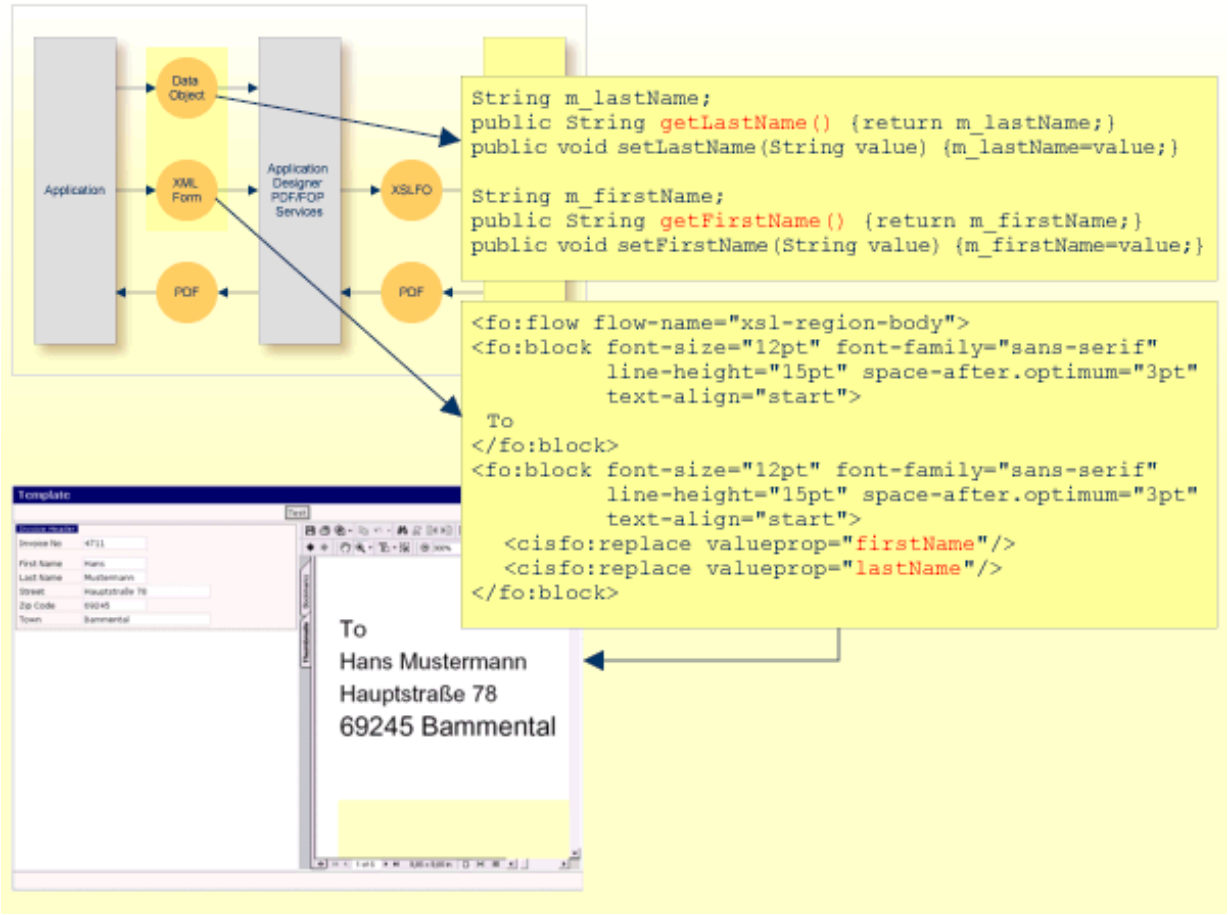
The server side PDF creation is what this document is about. Your server program uses the Application Designer APIs for creating PDF via FOP. The PDF files can be made available for the client (i.e. reachable by URL) and the client can integrate the files into its pages. Typically, the Adobe Reader is started on the client and the user can decide whether to print the document.

The server side printing is available by using a corresponding API. You can create an FOP document and call the print API. You have to define the printer on which you want to print.

Note: Server side printing requires JDK 1.4 to be used inside your application server. (JDK 1.3. only allows to print on the standard printer of your application server.)

Typical Example

The following image shows an example of how the framework is applied:



You see that the binding between the form's XML and the data object is the same as you know from the binding between Application Designer HTML pages and adapter objects. In fact, it is the same technology now transferred to PDF output scenarios.

The following topics are covered below:

- Data Object (Adapter)
- Ways of Usage

Data Object (Adapter)

At runtime, the data object provides the data that is filled into the form. The same rules that you already know from the Application Designer adapter objects also apply to the data object:

- Properties can be nested: `address.street` means that first `address` is accessed, then `street` within the address object.
- Properties can be implemented - at any level - by corresponding `set/get` methods and/or by implementing the interface `IDynamicAccess`.

For detailed information, see *Binding between Page and Adapter* in the *Java Pages Development*.

Ways of Usage

There are several ways of using the Application Designer PDF/FOP services.

- You may store the XML forms somewhere in the database or in the file system. By this you can, for example, allow your customer to set up own forms that bind to the same data objects but contain different rendering information (e.g. different logos, different order of columns).
- You may generate the XML form dynamically. If having a flexible grid of information, then the number of columns depends, for example, on certain parameters.

It is up to you to decide.

Browser Output of PDF Documents

After having created a PDF document, you typically want to output it somewhere in the browser. There are two questions that you have to consider:

- How do you store the generated PDF document in your server so that it is reachable by URL?
- How do you embed the document into Application Designer HTML pages?

The following topics are covered below:

Storing the Generated PDF

The Application Designer PDF APIs will generate a PDF byte stream. You have to store this byte stream in a certain way so that it can be reached by a URL from the browser. There are several ways to do so:

- You can store the document in the file system of the application server, for example, somewhere inside the file system belonging to your web application. However, this is a “quick and dirty” way and only works properly for some application servers (e.g. Tomcat, Jetty, etc.): it assumes that there is a file system in which your application is stored. Some application servers do not store their web applications in the file system at all, but hold them within databases.
- You can store the data in a way that it is not reachable by a “fixed URL to a file” but by a “soft, servlet-based” URL.

If using the second option and embedding PDF into Application Designer page processing, then Application Designer already provides a so-called “session buffer”. Inside the session buffer, you can store any object under a certain name. It will return a URL that you can pass to the browser as the URL to the stored content.

An example is provided in the section *The First PDF Document*.

Embedding the PDF into your Application Designer Page

- You can define a SUBPAGE control. A SUBPAGE control is a frame that can be dynamically loaded with a URL. This means you can store the PDF result somewhere in the file system so that it is accessible via a URL. The URL is passed to the SUBPAGE control - and the PDF is shown inside the frame accordingly.
- You can define an own frame in your frameset arrangement and send the URL of the PDF document to this frame.

An example is provided in the section *The First PDF Document*.

97 The First PDF Document

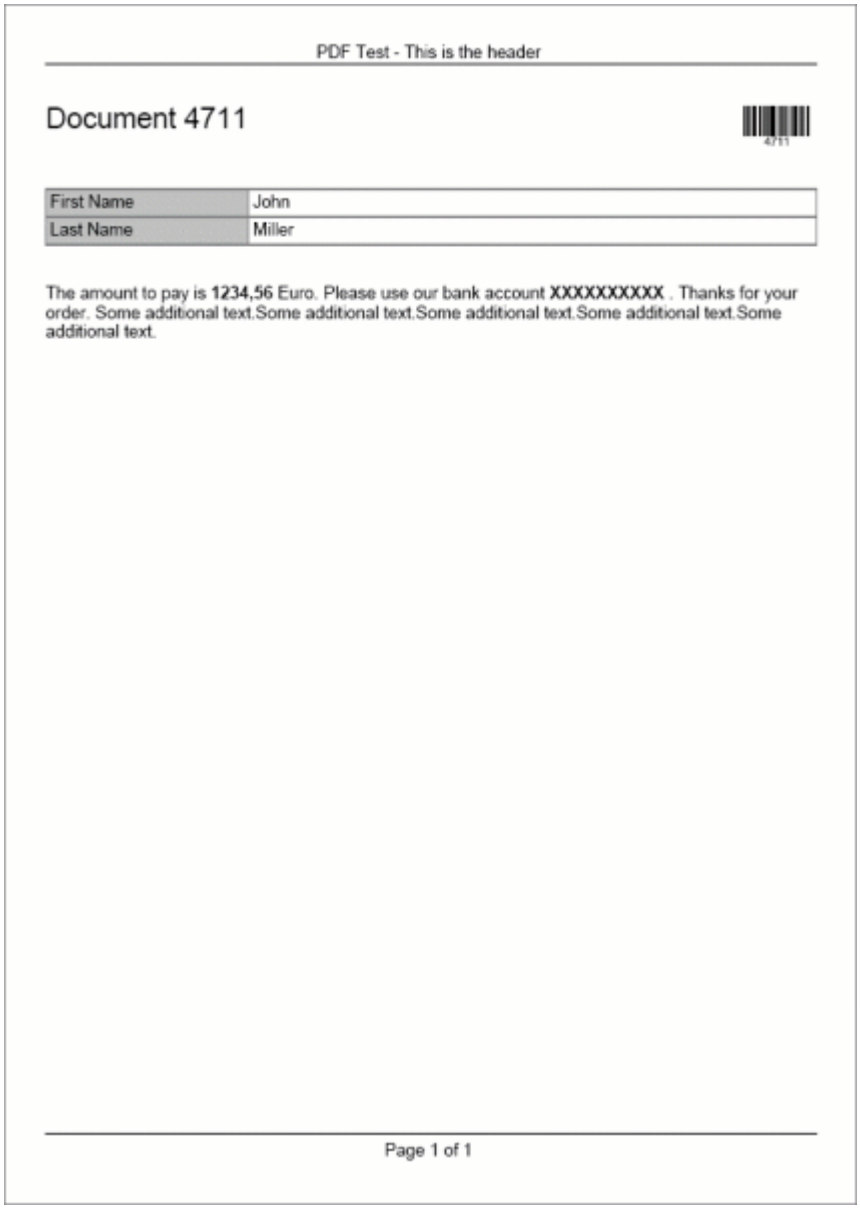
This chapter provides a brief tutorial in which you get to know all important steps for creating a PDF document that is rendered inside the browser. The tutorial describes how to do this using the development workplace. It is required that you have a basic understanding of Application Designer.

Overview of Required Steps

The following steps are required to create a PDF document:

1. Create the XML form.
2. Create the Java class that provides the data objects for the XML form.
3. Call the APIs to generate PDF.
4. Integrate the PDF into an Application Designer page.

The result of this tutorial will be the following document:



The document contains the following:

- Header, footer, body.
- Differently formatted texts.
- Barcode.
- Table of data.
- Some flat text containing dynamic data.

Creating the XML Form

Start the Application Designer development workplace (*http://localhost:51000/cis/HTML-BasedGUI/workplace/ide.html*) and open a project in the navigation frame (this tutorial uses the project "cisyourfirstproject").

Create a new layout. In the resulting dialog, enter a name (for example, "pdftest.xml") for the XML form definition and select the **layout template** for the PDF output. A new layout will be opened and preloaded with template information.

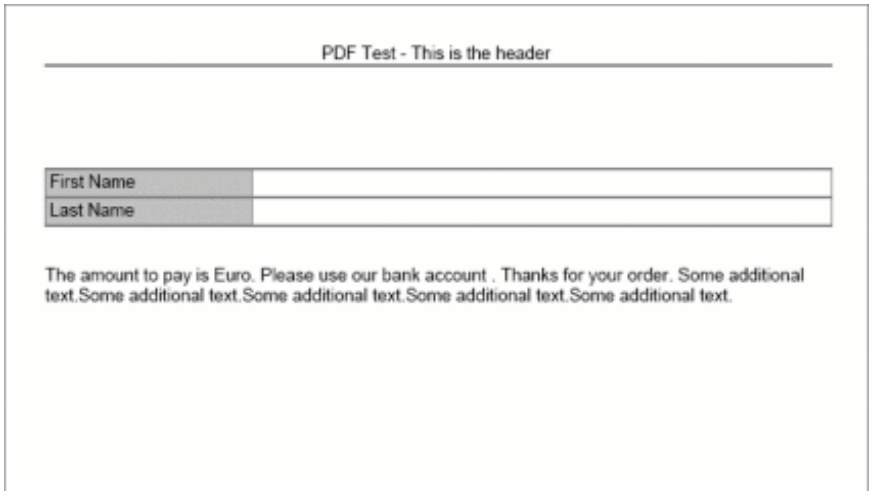
The Layout Painter is now used in the same way as you normally use it for creating Application Designer HTML pages. When you **preview** the layout in the Layout Painter, Adobe Reader is opened in the preview area, displaying the current XML form definition.

By adding controls into the template's control structure, you can modify the rendering result accordingly. Use the editor functions to create the following XML form definition:

```
<cisfo:foppage2 objectclass="PDFDataObject" pageheight="29.7cm" pagewidth="21cm" ↩
margintop="1cm" marginbottom="0.5cm" marginleft="1cm" marginright="1cm" ↩
headerheight="1.5cm" footerheight="1.3cm">
    <cisfo:styleclasses2>
        <cisfo:style2 stylename="THICK" fontsize="20pt" fontstyle="normal">
        </cisfo:style2>
        <cisfo:style2 stylename="LABEL" backgroundcolor="#C0C0C0" ↩
bordercolor="#808080" borderstyle="solid" borderwidth="0.1mm" padding="1mm">
        </cisfo:style2>
        <cisfo:style2 stylename="DEFAULTOUT" bordercolor="#808080" ↩
borderstyle="solid" borderwidth="0.1mm" padding="1mm">
        </cisfo:style2>
    </cisfo:styleclasses2>
    <cisfo:header2>
       <cisfo:rowtextblock2 text="PDF Test - This is the header" textalign="center">
        </cisfo:rowtextblock2>
        <cisfo:hline2>
        </cisfo:hline2>
    </cisfo:header2>
    <cisfo:footer2>
        <cisfo:hline2>
        </cisfo:hline2>
        <cisfo:vdist2 height="2mm">
        </cisfo:vdist2>
        <cisfo:rowtextblock2 text="Page" textalign="center">
            <cisfo:pagenumber2 separator=" of ">
            </cisfo:pagenumber2>
        </cisfo:rowtextblock2>
    </cisfo:footer2>
    <cisfo:body2>
        <cisfo:rowtable2 columnwidths="17cm; 2cm">
```

```
            <cisfo:row2>
                <cisfo:replace2 valueprop="docNumberText" stylename="THICK">
                </cisfo:replace2>
                <cisfo:barcode2 valueprop="docNumber" barcodetype="CODE39" ↩
height="10">
                </cisfo:barcode2>
            </cisfo:row2>
        </cisfo:rowtable2>
        <cisfo:vdist2 height="10mm">
        </cisfo:vdist2>
        <cisfo:rowtable2 columnwidths="5cm; 14cm">
            <cisfo:row2>
                <cisfo:celltext2 text="First Name" stylename="LABEL">
                </cisfo:celltext2>
                <cisfo:replace2 valueprop="firstName" stylename="DEFAULTOUT">
                </cisfo:replace2>
            </cisfo:row2>
            <cisfo:row2>
                <cisfo:celltext2 text="Last Name" stylename="LABEL">
                </cisfo:celltext2>
                <cisfo:replace2 valueprop="lastName" stylename="DEFAULTOUT">
                </cisfo:replace2>
            </cisfo:row2>
        </cisfo:rowtable2>
        <cisfo:vdist2 height="10mm">
        </cisfo:vdist2>
        <cisfo:rowtextblock2>
            <cisfo:text2 text="The amount to pay is ">
            </cisfo:text2>
            <cisfo:textreplace2 valueprop="amount" fontweight="bold">
            </cisfo:textreplace2>
            <cisfo:text2 text=" Euro. Please use our bank account ">
            </cisfo:text2>
            <cisfo:textreplace2 valueprop="account" fontweight="bold">
            </cisfo:textreplace2>
           <cisfo:text2 text=". Thanks for your order. Some additional text.Some ↩
additional text.Some additional text.Some additional text.Some additional text.">
            </cisfo:text2>
        </cisfo:rowtextblock2>
    </cisfo:body2>
</cisfo:foppage2>
```

After having entered all this (you can also cut and paste the XML from this documentation into your XML document) and when previewing the page, the top of the page should look as follows:

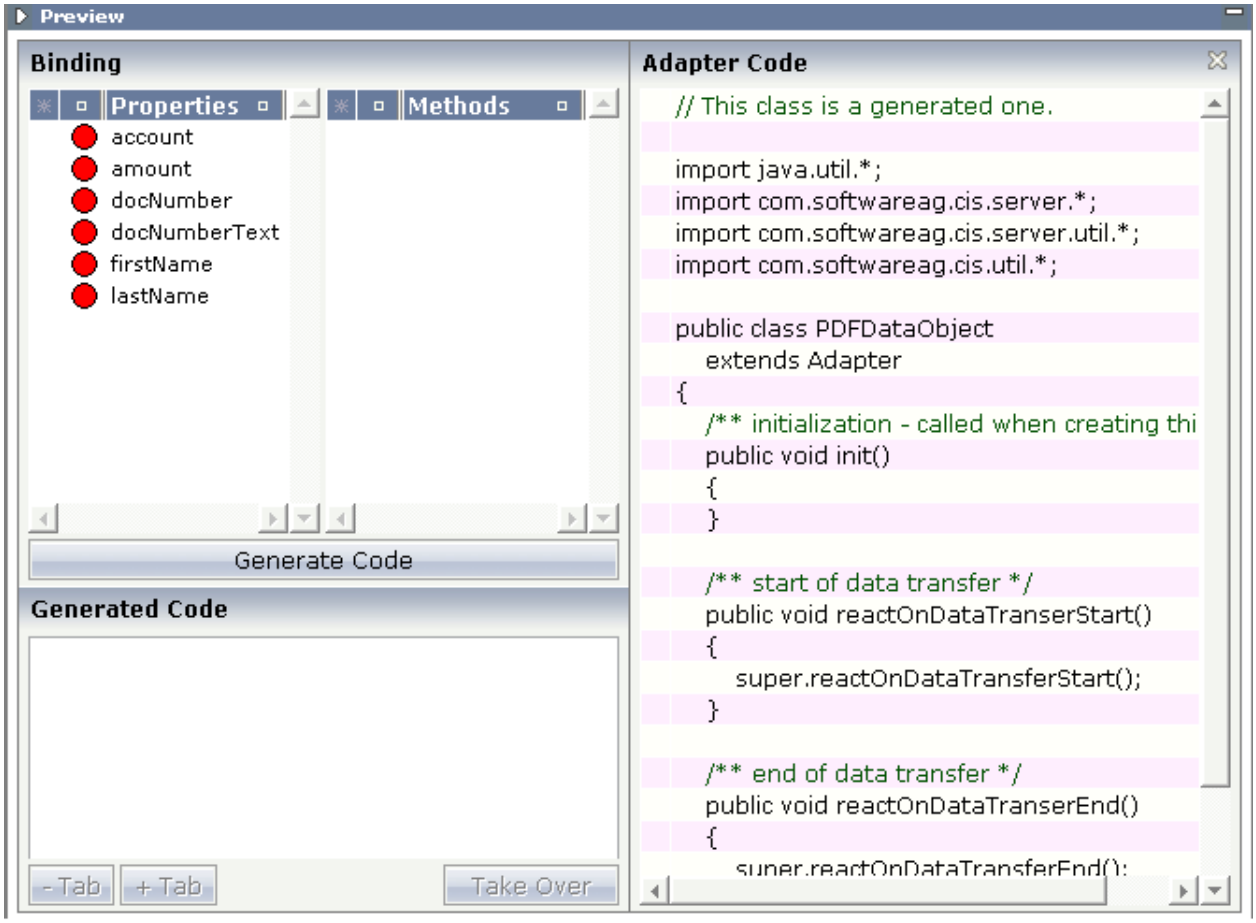


The document structure is already visible, but some data is still missing - the data that is provided by a corresponding data object.

Creating the Java Class

In the first tag of the XML form (CISFO:FOPPAGE2) there is a property `objectclass`. This is the class that provides the data that will be mixed into the document. In the example, the name of this class is `PDFDataObject`.

In order to create the class, you may use the **Code Assistant** which is part of the Layout Painter:



You should create some Java code that looks as follows:

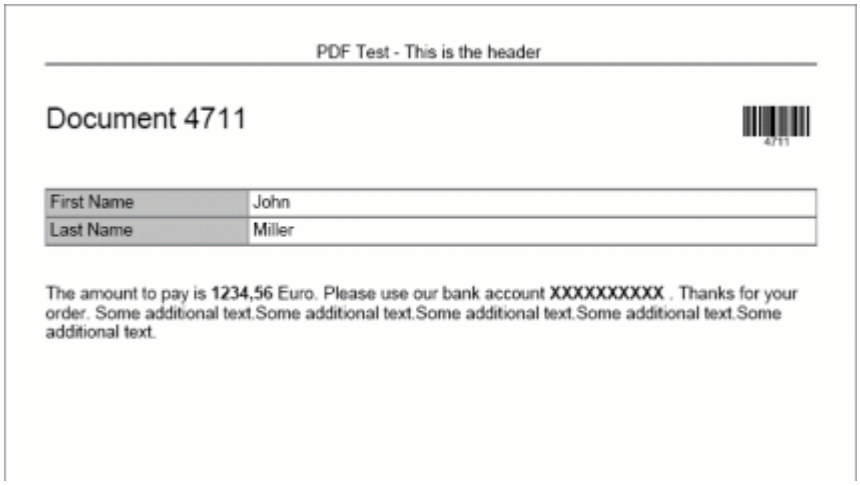
```
import java.util.*;

import com.softwareag.cis.context.SessionContext;
import com.softwareag.cis.print.fop.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

// class for PDF creation with CISFO2 tags

public class PDFDataObject
    implements IFOPDataObject
{
    // -------------------------------------------------------------------------
    // constructor
    // -------------------------------------------------------------------------

    // default constructor
    public PDFDataObject()
    {
    }
```

```
// Called if the corresponding print form is previewed within the Layout Painter
public PDFDataObject(PDFFOPPreviewOnly previewObject)
{
    m_docNumber = "4711";
    m_firstName = "John";
    m_lastName = "Miller";
    m_account = "XXXXXXXXXX";
    m_amount = "1234,56";
}

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

// property >account<
String m_account;
public String getAccount() { return m_account; }
public void setAccount(String value) { m_account = value; }

// property >amount<
String m_amount;
public String getAmount() { return m_amount; }
public void setAmount(String value) { m_amount = value; }

// property >firstName<
String m_firstName;
public String getFirstName() { return m_firstName; }
public void setFirstName(String value) { m_firstName = value; }

// property >lastName<
String m_lastName;
public String getLastName() { return m_lastName; }
public void setLastName(String value) { m_lastName = value; }

// property >docNumberText<
public String getDocNumberText() { return "Document " + m_docNumber; }

// property >docNumber<
String m_docNumber;
public String getDocNumber() { return m_docNumber; }
public void setDocNumber(String value) { m_docNumber = value; }

// ------------------------------------------------------------------------
// Internationalization for this data object
// ------------------------------------------------------------------------

String m_dateFormat = SessionContext.DATE_MMSDDSYYYY;
public String getDateFormat() { return m_dateFormat; }

String m_dayTimeFormat = SessionContext.TIME_HHCMMCSS;
public String getDayTimeFormat() { return m_dayTimeFormat; }
```

```
    String m_decimalSeparator = SessionContext.DEC_SEPARATOR_POINT;
    public String getDecimalSeparator() { return m_decimalSeparator; }

    String m_language = "E";
    public String getLanguage() { return m_language; }

    String m_timeStampFormat = SessionContext.DATE_MMSDDSYYYY;
    public String getTimeStampFormat() { return m_timeStampFormat; }


}
```

Compile this code so that the generated class is available in the directory *…/cisyourfirstproject/app-classes/classes*.

When now previewing the page, the top of the page should look as follows:



Let us have a closer look at the code:

- It implements an interface `com.softwareag.cis.print.fop.IFOPDataObject`. This interface provides some methods that the Application Designer PDF/FOP framework requires in order to provide internationalisation support.
- It provides the properties that are required by the page.
- It provides a special constructor by which the properties are filled with some dummy data.

The special constructor is used by Application Designer when previewing the form in the Layout Manager. (Later on you will - as part of your programs - fill the data object inside your application and pass it to the PDF/FOP processing together with the form.)

Calling the APIs at Runtime

In the previous steps you created:

- an XML form (stored in an XML file within a certain Application Designer project),
- a Java class representing the net data behind the XML form.

Creating a PDF file now is quite simple: you just have to load the XML form and create an instance of the data object, and pass both to the PDF/FOP processor framework. The code might look like:

```
String xmlForm = WebResourceReader.readTextFileIntoStringWithLinebreak
      (findServletContext(),"/cisyourfirstproject/xml/pdftest.xml");
 PDFDataObject pdo = new PDFDataObject();
 pdo.setDocNumber("4711");
 pdo.setAccount("1234567890");
 pdo.setFirstName(m_firstName);
 pdo.setLastName(m_lastName);
 pdo.setAmount(m_amount);
 IPDFFOPService pdffop = PDFFOPServiceFactory.createPDFFOPService(xmlForm,pdo);
 byte[] pdf = pdffop.generatePDF();
```

In the code, the XML file is read by using a `WebResourceReader` class which is part of the Application Designer runtime environment. This class uses the servlet context's mechanism to access files that are part of your web application. As a consequence, the file is accessed in such a way that is compatible to any cluster structures of the application server you might use. Of course, you can also read the XML form in any other way. Maybe you store your forms inside a database and read the data from there.

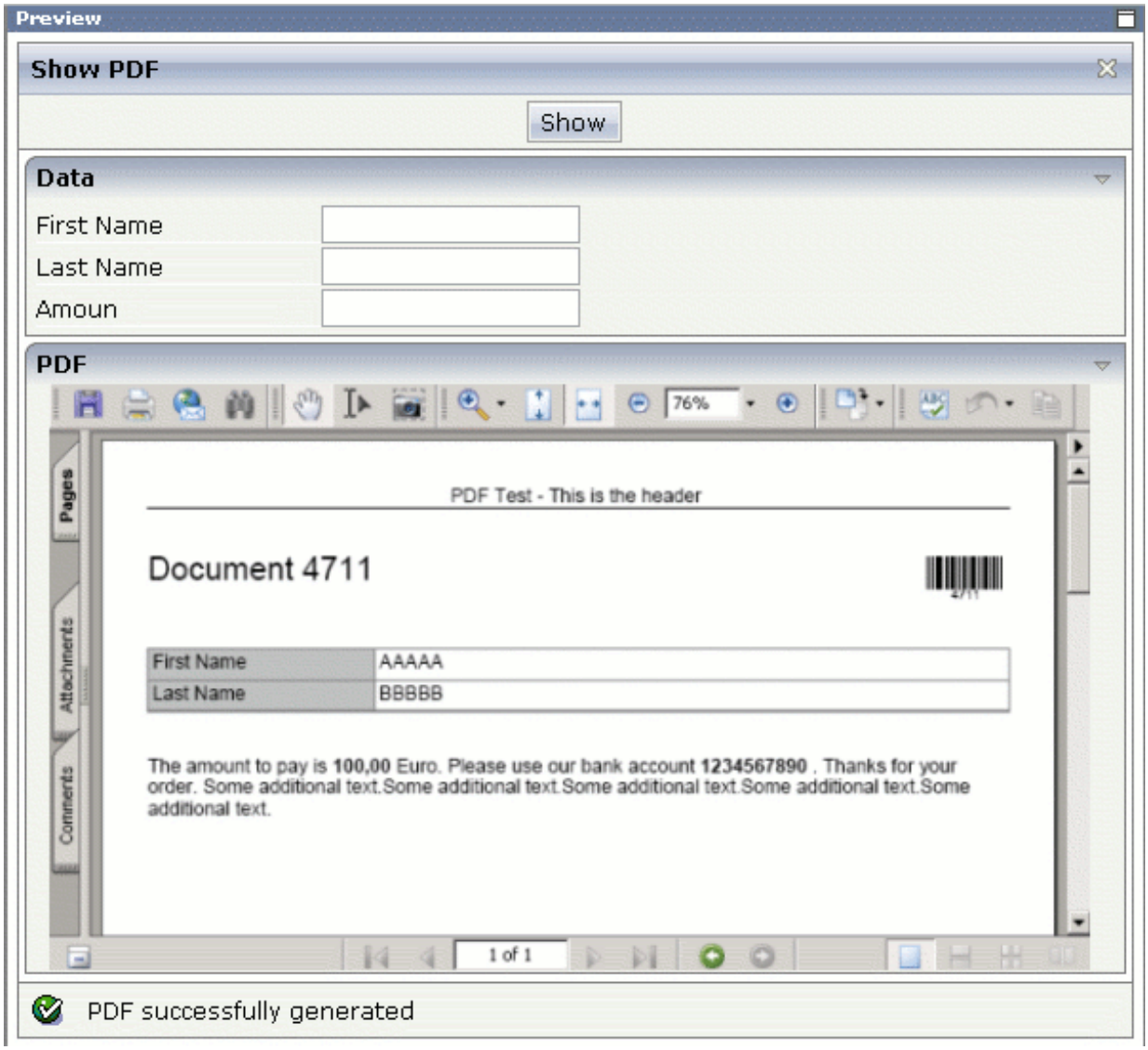
The data object is simply created and filled with certain data. Part of the data are passed as fixed literals, other parts are dynamically passed by corresponding member variables.

The PDF processor is created by calling the `PDFFOPServiceFactory` class and requesting an instance. Finally, the instance is invoked and PDF is passed back as a byte array.

It is important to note that in principle the creation of PDF is completely independent from the normal GUI adapter processing inside the Application Designer; i.e. you can use it within your application at any point - not only inside Application Designer adapter objects that represent Application Designer HTML pages.

Integrating the PDF into an Application Designer Page

Now, let us build an Application Designer HTML page that looks as follows:



You can enter some parameters and choose the **Show** button. As a result, the corresponding PDF page will be displayed as part of the page.

The layout of the page looks as follows:

```
<page model="ShowPDFAdapter">
    <titlebar name="Show PDF">
    </titlebar>
    <header withdistance="false">
        <button name="Show" method="onShow">
        </button>
    </header>
    <pagebody takefullheight="true">
        <rowarea name="Data">
            <itr>
                <label name="First Name" width="120">
                </label>
                <field valueprop="firstName" width="200">
                </field>
            </itr>
            <itr>
                <label name="Last Name" width="120">
                </label>
                <field valueprop="lastName" width="200">
                </field>
            </itr>
            <itr>
                <label name="Amount" width="120">
                </label>
                <field valueprop="amount" width="200">
                </field>
            </itr>
        </rowarea>
        <rowarea name="PDF" height="100%">
            <itr width="100%" height="100%" fixlayout="true">
                <subpage valueprop="pdfURL" height="100%" width="100%">
                </subpage>
            </itr>
        </rowarea>
        <vdist>
        </vdist>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

There are three fields to do the input and there is one SUBPAGE control which renders a URL that is dynamically passed by the property `pdfURL`.

Let us have a look at the corresponding adapter class code:

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

import java.util.*;

import com.softwareag.cis.print.fop.IPDFFOPService;
import com.softwareag.cis.print.fop.PDFFOPServiceFactory;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class ShowPDFAdapter
    extends Model
{
    // -----------------------------------------------------------------------
    // property access
    // -----------------------------------------------------------------------

    // property >amount<
    String m_amount;
    public String getAmount() { return m_amount; }
    public void setAmount(String value) { m_amount = value; }

    // property >firstName<
    String m_firstName;
    public String getFirstName() { return m_firstName; }
    public void setFirstName(String value) { m_firstName = value; }

    // property >lastName<
    String m_lastName;
    public String getLastName() { return m_lastName; }
    public void setLastName(String value) { m_lastName = value; }

    // property >pdfURL<
    String m_pdfURL = "about:blank";
    public String getPdfURL() { return m_pdfURL; }
    public void setPdfURL(String value) { m_pdfURL = value; }

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

    public void onShow()
    {
     try
 {
      // build pdf
  String xmlForm = WebResourceReader.readTextFileIntoStringWithLinebreak
                   (findServletContext(),"/cisyourfirstproject/xml/pdftest.xml");
  PDFDataObject pdo = new PDFDataObject();
  pdo.setDocNumber("4711");
  pdo.setAccount("1234567890");
  pdo.setFirstName(m_firstName);
```

```
  pdo.setLastName(m_lastName);
  pdo.setAmount(m_amount);
  IPDFFOPService pdffop = PDFFOPServiceFactory.createPDFFOPService(xmlForm,pdo);
  byte[] pdf = pdffop.generatePDF();
  // build dynamic URL
  m_pdfURL = findCISSessionContext().getSessionBuffer().addPDF("test" + (new ↩
Date()).getTime(),pdf);
  outputMessage(MT_SUCCESS,"PDF successfully generated");
 }
     catch (Throwable e)
 {
      outputMessage(MT_ERROR,e.toString());
 }
    }
}
```

The critical method is the `onShow` method that is invoked when the user chooses the **Show** button. In the method, the API is called and a PDF byte array is created as mentioned in the previous section.

The result is not stored in the file system but is passed to a so-called “session buffer” that is made available by Application Designer. The session buffer is a transient store for temporary data that is made available to the browser via a URL. The data you pass to the session store is passed by using an API; in the API, you specify a certain name for the data. The URL that is passed back to the application can be used inside the browser to access the transient data. Internally, the URL is a link to certain servlet provided by Application Designer that accesses the transient data.

As a consequence, the PDF bytes are not stored in the file system in order to be displayed inside the browser. This particularly means that your application is able to be operated in a clustered environment.

Note: Of course, there may be different scenarios in your environment: maybe you have one central file server which also serves as a web server, and you park all your PDF documents on this server. In this case, you do not need the session buffer.

The session buffer is automatically removed when the user's session ends. If you have long-lasting user sessions, make sure that the information in the session buffer is removed when it is not required anymore. Removing is done by using the session buffer's Java API.

98 Printing

There are two commonly-used ways for producing paper printouts:

- Create PDF via PDF/FOP processing and make it available in the user's browser. This is what the example in the section *The First PDF Document* does.
- Directly print FOP to a server side printer. This is what this section is about.

Server side printing is done via the interface `IPDFFOPService`:

```
public interface IPDFFOPService
        {
        public String generateXSLFO();
        public byte[] generatePDF();
        public void print(String printerName) throws FOPPrintException;
        public byte[] generatePDFfromFOXML();
        public ILog getLog();
        public Protocol getProtocol();
        public String getTranslationFileName();
        }
```

By using the method `print(...)`, you can send a document directly to a printer.