

Application Designer

Special Development Topics

Version 9.1.2

October 2019

This document applies to Application Designer Version 9.1.2 and all subsequent releases.

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

Copyright © 2005-2019 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: CIT-DEVTOPICS-912-20190816

Table of Contents

Preface	ix
1 About this Documentation	1
Document Conventions	2
Online Information and Support	2
Data Protection	3
I Controls for Absolute Positioning	5
2 Example	7
3 Controls	11
4 ABSFOLDER	13
Properties	14
5 ABSFIELD	15
Properties	16
6 ABSICON	19
Properties	20
7 ABSDYNICON	21
Example: Moving an Icon	22
Properties	25
8 ABSTEXTOUT	27
Properties	28
9 ABSLABEL	31
Properties	32
10 ABSTABLE0/ABSTR	35
ABSTABLE0 Properties	37
ABSTR Properties	39
11 ROWABSAREA	41
All Controls Directly Inside the Page Tag	42
All Controls Inside the Page Body	42
Explicit Areas for Absolute Positioning	44
Properties	45
12 ABSAREA	47
Example	48
Properties	49
II Writing Reports	51
13 Introduction	53
14 Writing Reports by Using the REPORT Control	57
The Very Beginning - A White Report Area	58
Rendering a Grid into the REPORT Control	59
Using Special Styles for Cell Output	61
Adding Some Text	62
Adding a Second Grid	63
Adding an Image	65
HTML Rendering - PDF Rendering	66
Reacting on Mouse Clicks	67

REPORTInfo API	69
REPORT Controls versus TEXTGRID Control	70
Properties	70
15 Creating Statistical Charts	73
Structure	74
Class com.softwareag.cis.chart.CHARTInfo	75
Creating a Simple Chart	75
Setting the Scale of the y-Axis	78
16 Embedding Statistical Charts into Reports	81
Creating an SVG Graphic and Embedding it into a Report	82
Creating a JPEG Graphic and Embedding it into a Report	83
Pay Attention when Sizing your Graphic	84
17 Using the Special Chart Control QUADRANTPLOT	85
Simple Example	86
Properties	89
18 Creating Simple Charts Quickly Using the PIVOT Control	91
Simple Example	92
Properties	95
19 Best Practice Hints	97
III Non-Visual Controls and Hot Keys	99
20 AUTOCOMPLETE	101
General Information	102
Example	103
Data Sources for Populating the Values	103
Properties	106
21 TIMER	109
Example	110
Properties	112
22 Extended Hot Key Management	113
Direct Hot Key Definitions with Certain Controls	114
Hot Key Definitions for Certain Controls	114
IV Binding between Page and Adapter	117
23 Phases of Adapter Processing	119
SET/INVOKE/GET Phase - The Default Phases	120
INIT Phase when Adapter is Constructed	121
DESTROY Phase when Adapter is Deregistered	122
24 Class Binding	123
Direct Class Binding	124
Generic Class Binding	125
25 Types of Property Binding	127
26 Java Bean Property Binding	129
Class Binding	130
Method Binding	130
Property Binding	131
Access Path Restrictions	134

27 Dynamic Access Property Binding	137
Interface IDynamicAccess	138
Example	138
28 XML Property Binding	143
29 Getting Information about Access Paths	145
30 Exception Management Inside an Adapter Object	149
Normal Exceptions are to be Handled by the Application	150
Errors and Runtime Exceptions - The Default Behavior	151
Interrupting the Application Designer Request Processing - AdapterNotAvailableError	152
Errors and Runtime Exceptions - The Special Behavior	317
31 Additional Interfaces	155
Extending the Set of Simple Data Types	156
Avoid the Getting of Certain Simple Data Type Properties	157
Exchanging Objects by Converter Objects	158
V Details on Session Management	159
32 HTTP Sessions - Application Designer Sessions	161
33 Application Designer Session - Application Designer Subsessions	163
34 Application Designer Subsession - Application Designer Adapter Objects	165
35 How Things Start	167
Starting an Application Designer Session	168
Starting Additional Application Designer Subsessions	168
36 How Things End	171
End of an Application Designer Session	172
End of an Application Designer Subsession	172
End of an Application Designer Adapter	172
37 Workplace Management	173
38 Saving Context Data	175
Different Levels of Context	176
Accessing the Context	176
Typical Usage Scenarios	367
39 Session IDs	179
VI Application Project Management	181
40 What is an Application Project?	183
41 Class Loading Issues	185
42 Application Project Directory	187
43 Application Project Context Root	189
44 Creating an Application Project	191
45 Tools for Application Project Management	193
VII Dynamic Page Layout	195
46 Introduction	197
47 Scenarios	199
48 Dynamic Pages - Normal Pages	201
49 Programming Dynamic Pages	203
50 Interface IDynamicPageMgmt	207

51 Background Information	209
Link to Session Management	210
Performance Considerations	211
URL Position of the Pages	211
Dynamic Pages - Multi Language Management	212
52 Dynamic Pages - Dynamic Adapters	213
VIII Becoming a Member of the Startup Process	215
53 Overview	217
54 Startup Class	219
55 Registration	221
IX Adapting the Look & Feel	223
56 Introduction	225
57 Style Sheet File	227
58 Writing a New Style Sheet File	229
59 Selecting the Right Style Sheet	231
60 Dynamic Selection of the Style Sheet File	233
What You Can Do	234
Example	234
61 Static Selection of the Style Sheet File	237
X Controls for Database Reporting	239
62 Basics	241
Two Types of DB Controls	242
When to Use Which Type	243
63 DBQUERY	245
Example	247
DBQUERY Properties	251
DBFILTER Properties	255
DBCOLUMN Properties	257
DBPARAMSINGLEVALUE Properties	259
DBPARAMDOUBLEVALUE Properties	260
Variant Management	261
PDF Generation	262
64 DBFIELD	263
Example	264
Properties	269
65 DBCOMBO	271
Example	272
Properties	275
66 DBSELECTOPTION	279
Example	280
Properties	284
67 DBCHECKBOX	287
Example	288
Properties	290
68 DBRADIOBUTTON	293

Example	294
Properties	297
XI Personalization of Pages	299
69 Goal	301
70 Customized Layout - Concepts	303
Overview	304
Dynamic Controls	306
Using Filters	306
Personalization Filter	307
Personalization Scenario Sequence	308
Maintaining Personalization Data	308
Persisting Personalization Data	308
71 Customized Layout - Example	309
XML Layout	310
Java Adapter Code	312
72 Customized Proposals - Concepts	315
Overview	316
Properties Used for Proposals	318
Personalization Scenario, Personalization Scenario Sequence	318
73 Customized Proposals - Example	319
XML Layout	320
Java Adapter Code	321
Directly Accessing Proposal Values	322
XII	323
74 Security Aspects	325
75 Portal Integration	329
Integrating Pages as Portlets	330
Session Management and Portlet API Support	332
Portlet Integration and AJAX	333
76 Using Layout Painter Extensions	335
Example	336
Details on the Extension	337
Extension Meets Pattern	347
77 Microsoft Silverlight Integration	353
Example	354
Implementation of the Sample Page	355
Integration of Silverlight	358
78 Integrating Application Designer Controls in HTML Pages	375
Example	376
Details on the Implementation	378
Invoking the Page in the Browser	379
PGHEAD Properties	380
PGCONTAINER Properties	381
79 Automated Testing	385

Preface

The information in this documentation is organized in the following parts:

Controls for Absolute Positioning	How to use absolute positioning for controls.
Writing Reports	Shows how Application Designer supports reporting.
Non-Visual Controls and Hot Keys	How to develop controls that do not have visual effects.
Binding between Page and Adapter	Describes data transfer between pages and adapter.
Details on Session Management	Gives details about session management.
Application Project Management	Gives details about project management.
Dynamic Page Layout	Gives details about dynamic page layout.
Becoming a Member of the Startup Process	Shows you how to become a member of the startup process.
Adapting the Look & Feel	Shows you how to provide high quality controls by simply specifying tags inside a layout definition.
Controls for Database Reporting	Shows you a simple and flexible way to develop typical reporting papers for querying database content.
Personalization of Pages	Shows you how to provide for customizable pages.
Security Aspects	How to avoid security risks.
Portal Integration	How to integrate your application with a portlet.
Using Layout Painter Extensions	How to generate layout elements into an existing layout.
Microsoft Silverlight Integration	How to integrate Microsoft Silverlight controls into your pages.
Integrating Application Designer Controls in HTML Pages	How to use this “outside-in approach” to integrate Application Designer controls and functionality in standard HTML pages.
Automated Testing	How to use test tool IDs for automated tests.

1

About this Documentation

■ Document Conventions	2
■ Online Information and Support	2
■ Data Protection	3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Monospace font	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	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

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <http://documentation.softwareag.com>. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at <https://empower.softwareag.com/>.

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#).

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at <http://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

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

Controls for Absolute Positioning

There is a special set of controls available to position them by defining their absolute x- and y-coordinates. 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:

Example

Controls

ABSFOLDER

ABSFIELD

ABSICON

ABSDYNICON

ABSTEXTOUT

ABSLABEL

ABSTABLE0/ABSTR

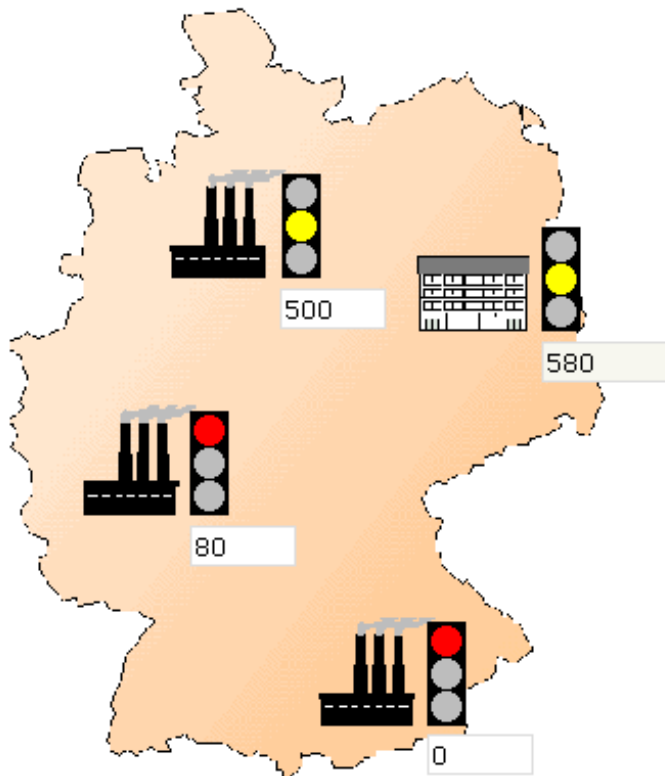
ROWABSAREA

ABSAREA

2 Example

A typical case for using absolute positioning is demonstrated in the following example:

XYZ Company



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.

3

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.

4 ABSFOLDER

■ Properties	14
--------------------	----

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	

5 ABSFIELD

■ Properties	16
--------------------	----

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	<p>Flushing behaviour of the input control.</p> <p>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.</p> <p>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 synchronization.</p> <p>Setting FLUSH to "screen" means that the changed value is populated inside the page. You</p>	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 representaion 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

	<p>then the input will be translated into "01.03.2004" (or other representation, dependent on date format settings).</p> <p>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.</p> <p>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.</p>		<p>xs:decimal</p> <p>xs:double</p> <p>xs:date</p> <p>xs:dateTime</p> <p>xs:time</p> <p>-----</p> <p>N n.n</p> <p>P n.n</p> <p>string n</p> <p>L</p> <p>xs:boolean</p> <p>xs:byte</p> <p>xs:short</p>
fieldstyle	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p> <p>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.</p>	Optional	
comment	<p>Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.</p>	Optional	

6 ABSICON

■ Properties	20
--------------------	----

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	

7

ABSDYNICON

▪ Example: Moving an Icon	22
▪ Properties	25

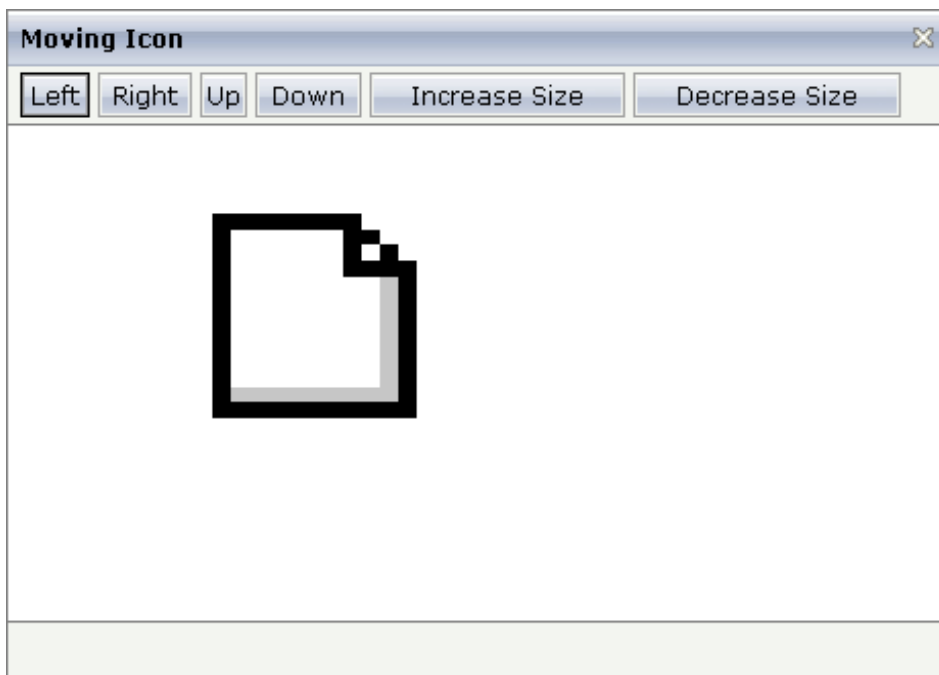
The ABSDYNICON is similar to the [ABSICON](#) control. Main difference: whereas the name of the image is defined statically by the ABSICON control, it is derived from an adapter property by the ABSDYNICON control. The image can be changed by the application. The ABSDYNICON is used to display the dynamic 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 ABSDYNICON 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.

8 ABSTEXTOUT

■ Properties	28
--------------------	----

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	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p> <p>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.</p>	Optional	
comment	<p>Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.</p>	Optional	

9

ABSLABEL

■ Properties	32
--------------------	----

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	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p>	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.

10

ABSTABLE0/ABSTR

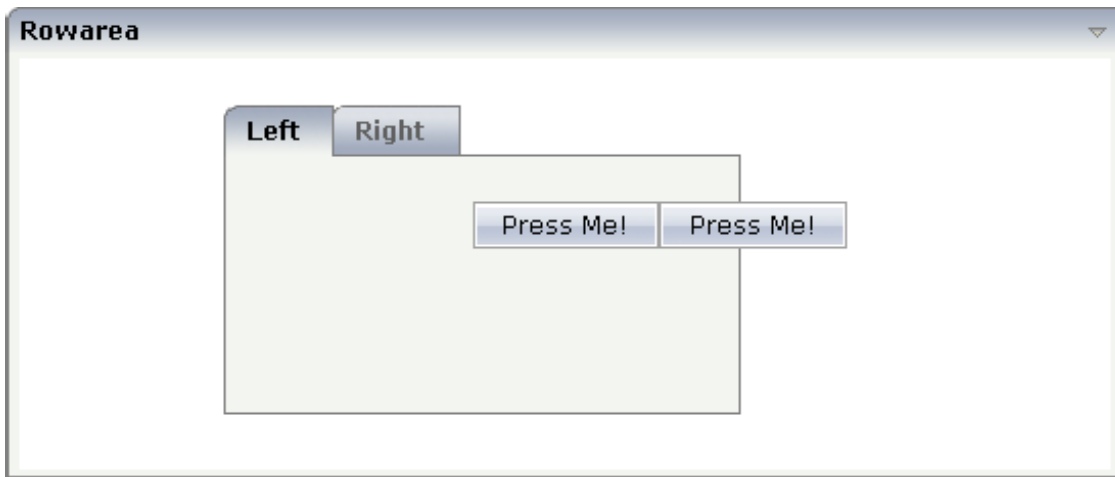
■ ABSTABLE0 Properties	37
■ ABSTR Properties	39

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

	<p>vertical sizing at all - or you use vertical pixel sizing for your controls - the property must be switched to 'false'.</p> <p>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.</p>		
fixlayout	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>	Optional	true false
tablestyle	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p> <p>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.</p>	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	<p>Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.</p>	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	

11

ROWABSAREA

■ All Controls Directly Inside the Page Tag	42
■ All Controls Inside the Page Body	42
■ Explicit Areas for Absolute Positioning	44
■ Properties	45

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>
```

```
</rowabsarea>  
</pagebody>
```

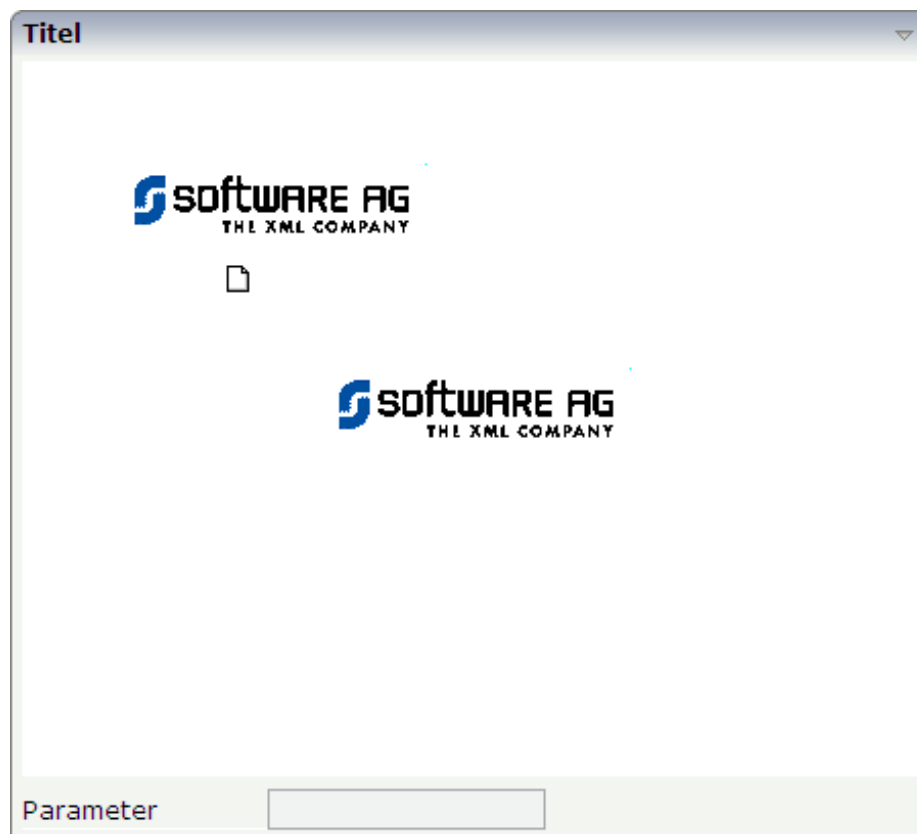
The controls are positioned relative to the page body's coordinates. I.e. the coordinate "(0;0)" is the left-top corner of the page body.

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:		150
	(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.		200
			250
	(B) Pixel sizing: just input a number value (e.g. "20").		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	

12

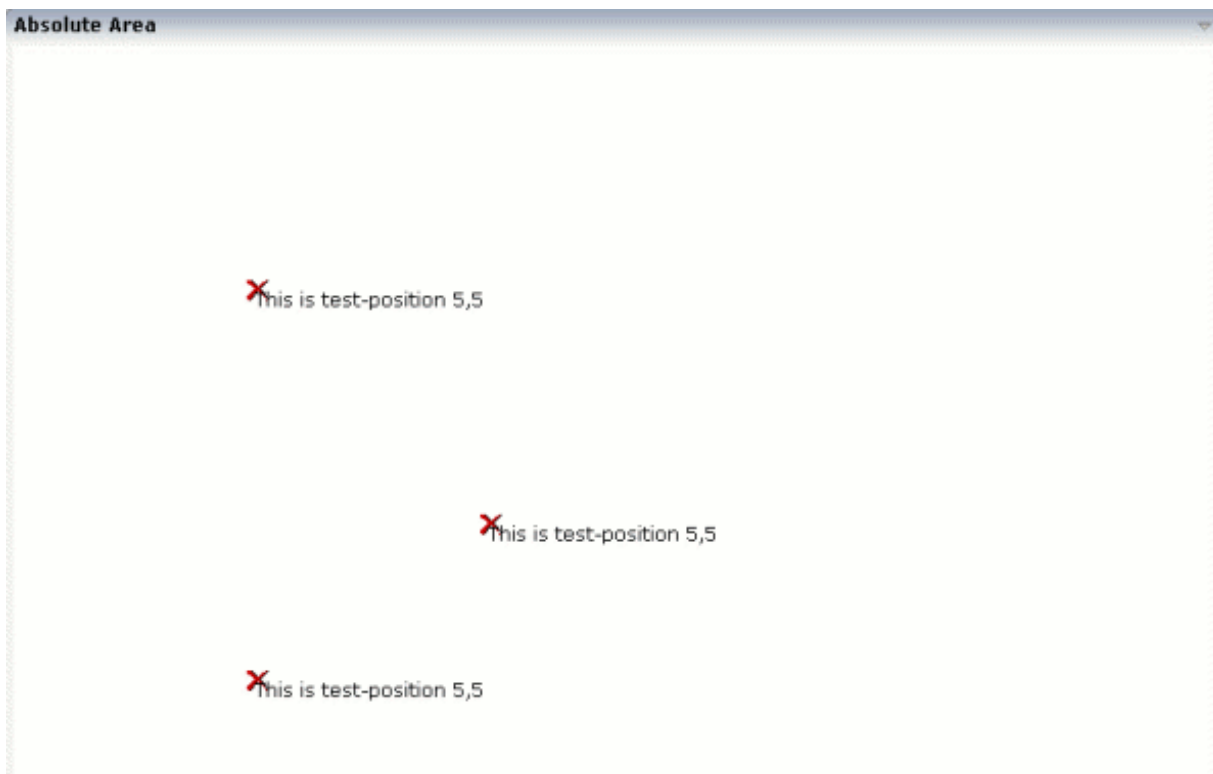
ABSAREA

■ Example	48
■ Properties	49

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.

II Writing Reports

This part describes how to create reports with Application Designer.

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

Introduction

Writing Reports by Using the REPORT Control

Creating Statistical Charts

Embedding Statistical Charts into Reports

Using the Special Chart Control QUADRANTPLOT

Creating Simple Charts Quickly Using the PIVOT Control

Best Practice Hints

When using the REPORT control with an embedded chart, the transformation from SVG to JPEG, GIF and other formats is only possible when the environment variable `DISPLAY` has been set. Example:

```
DISPLAY=:0.0
export DISPLAY
```


13

Introduction

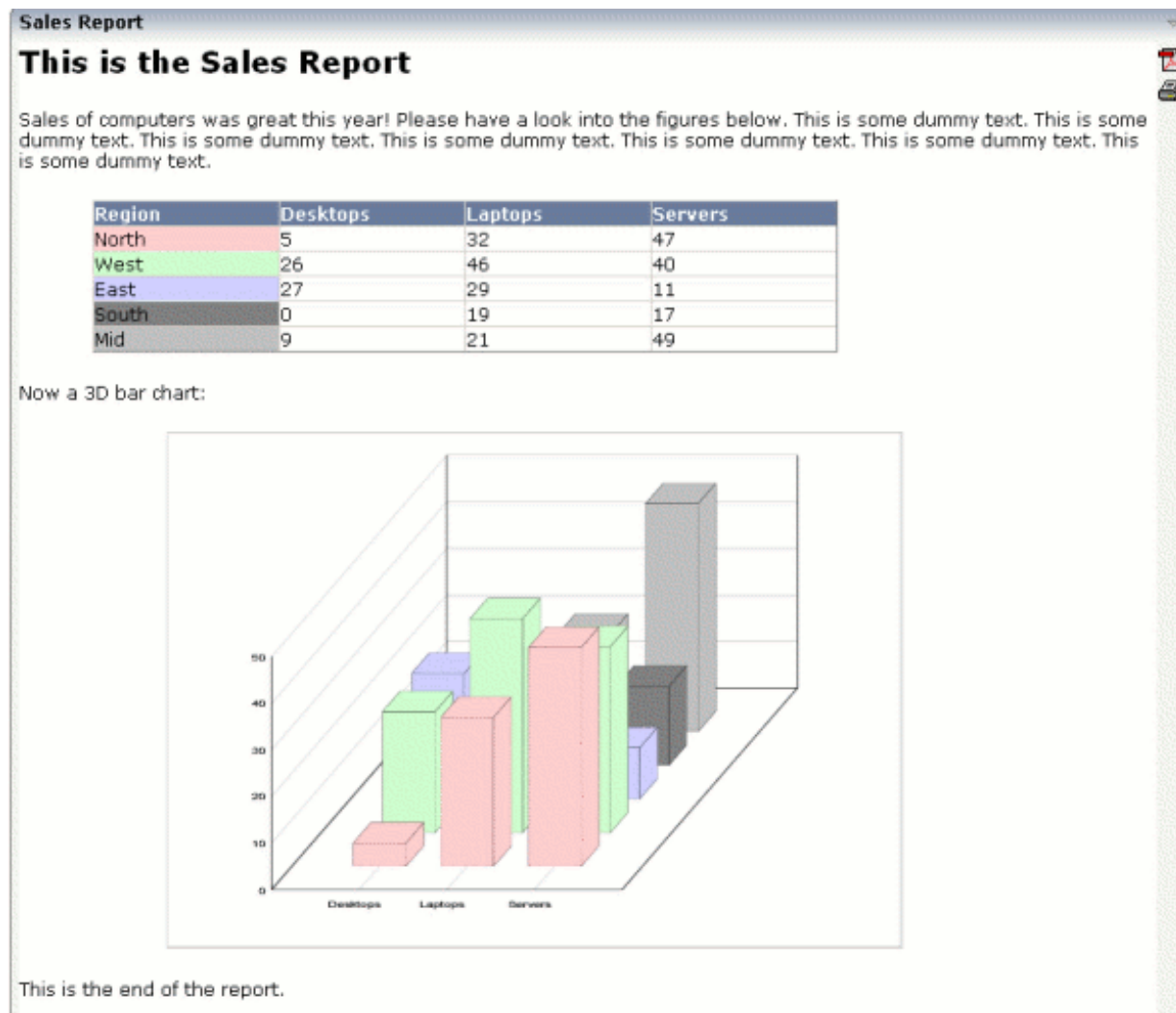
Application Designer offers flexible, easy-to-use controls and server side Java APIs to support reporting. Reporting functions include:

- Simple reporting document output, including
 - output of tables
 - output of text
 - output of statistical graphics
- The reporting output is automatically rendered into HTML and/or PDF without developers to take care of.

In addition, Application Designer provides a Java API to create a rich set of statistical charts. Functions in this area include:

- Various types of charts:
 - pie chart
 - bar chart
 - 3D bar chart
 - point chart
 - line chart
 - aggregation chart
- Different types of charts can be mixed - e.g. you can combine lines and bars inside one chart.
- Charts can be rendered into
 - SVG format (scalable vector graphics) - a W3C standard that is supported by up-to-date browsers
 - JPG format

While in principle Application Designer delivers independent APIs for the various issues (e.g. the charting API is independent from the reporting output API), all can be combined in order to write nice looking reports:



Reporting is not an island on its own but can be integrated easily in any Application Designer page, i.e. the output is done inside a normal Application Designer control that you place into an Application Designer page.

The objective of Application Designer reporting is:

- Create nice looking reports in an efficient way.
- Offer a report rendering API that can be flexibly used by your application logic frameworks.
- Have an automated PDF generation.
- Provide interactivity so that reports can be linked to other parts of your application.

In short: we do not provide a full-fledged reporting tool for SQL type of reporting - we offer something small and smart that is easy to integrate and to adapt into your application.

14

Writing Reports by Using the REPORT Control

■ The Very Beginning - A White Report Area	58
■ Rendering a Grid into the REPORT Control	59
■ Using Special Styles for Cell Output	61
■ Adding Some Text	62
■ Adding a Second Grid	63
■ Adding an Image	65
■ HTML Rendering - PDF Rendering	66
■ Reacting on Mouse Clicks	67
■ REPORTInfo API	69
■ REPORT Controls versus TEXTGRID Control	70
■ Properties	70

The REPORT control can be considered as a white sheet of paper in which you render reporting output. The output may consist of:

- grids
- text
- images (jpg/svg)

From the control definition point of view, the REPORT control is very easy: you just have to define the size of the control and a reference to an adapter property that represents the control on the server side. Let us start building up a simple report step by step.

The Very Beginning - A White Report Area

Let us first define the page layout:



The layout definition is:

```
<page model="Demo_Report_Adapter">
  <titlebar name="Demo Report">
  </titlebar>
  <pagebody takefullheight="true">
    <rowarea name="Demo Report" height="100%">
      <report reportprop="report" height="100%" showpdf="true">
      </report>
    </rowarea>
  </pagebody>
  <statusbar withdistance="false">
  </statusbar>
</page>
```

The adapter code is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.REPORTCellFormat;
import com.softwareag.cis.server.util.REPORTInfo;

public class Demo_Report_Adapter
  extends Adapter
{
  // property >report<
  REPORTInfo m_report = new REPORTInfo(this);
  public REPORTInfo getReport() { return m_report; }
}
```

On the layout side, you see the REPORT control referencing the property `report`. The property itself is of type `REPORTInfo`.

Please note: in the XML layout definition, the property `showpdf` is set to "true". If you do so, you also have to create the instance of `REPORTInfo` by calling the constructor in which the instance of the adapter object that manages the control is passed.

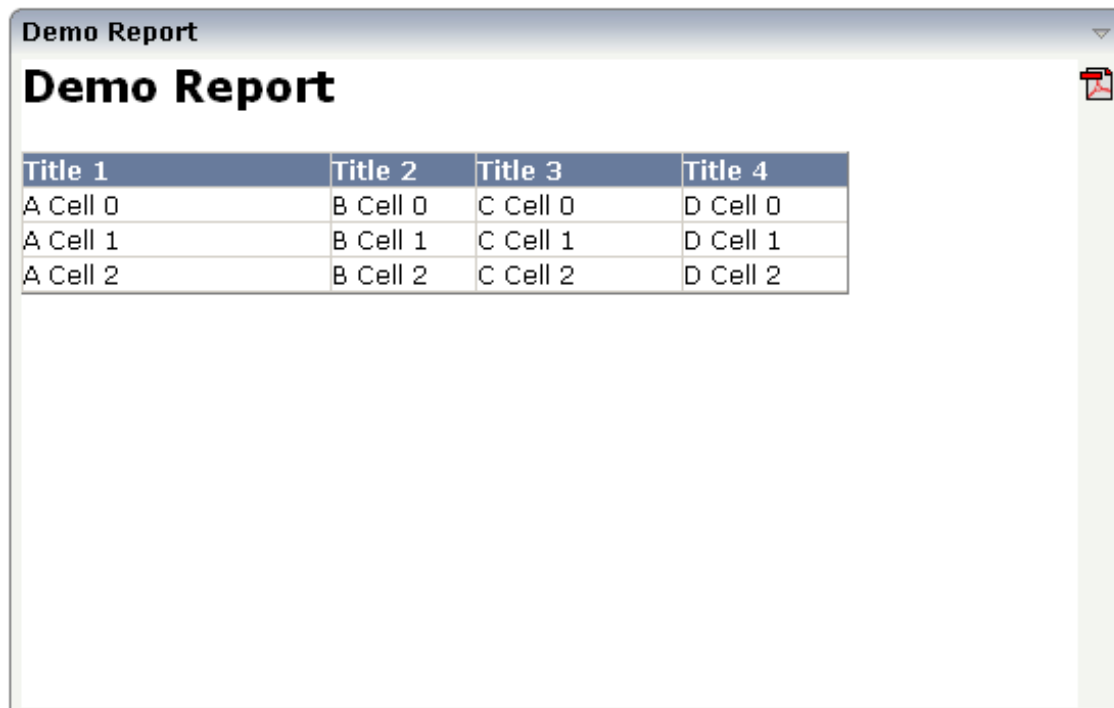
Rendering a Grid into the REPORT Control

The following extension is done to the adapter class:

```
public void init()
{
    renderGrid();
}

/**
 * Renders a grid with 4 columns and 3 data rows.
 */
private void renderGrid()
{
    m_report.addHeadlineCell("Title 1","150");
    m_report.addHeadlineCell("Title 2","70");
    m_report.addHeadlineCell("Title 3","100");
    m_report.addHeadlineCell("Title 4","80");
    m_report.addNewLine();
    for (int i=0; i<3; i++)
    {
        m_report.addCell("A Cell " + i);
        m_report.addCell("B Cell " + i);
        m_report.addCell("C Cell " + i);
        m_report.addCell("D Cell " + i);
        m_report.addNewLine();
    }
}
```

The control now looks as follows:



The screenshot shows a window titled "Demo Report" with a sub-header "Demo Report". Below the header is a table with 4 columns and 3 data rows. The columns are labeled "Title 1", "Title 2", "Title 3", and "Title 4". The data rows are labeled "A Cell 0", "A Cell 1", and "A Cell 2". The table is rendered with a blue header and a light gray background for the data rows.

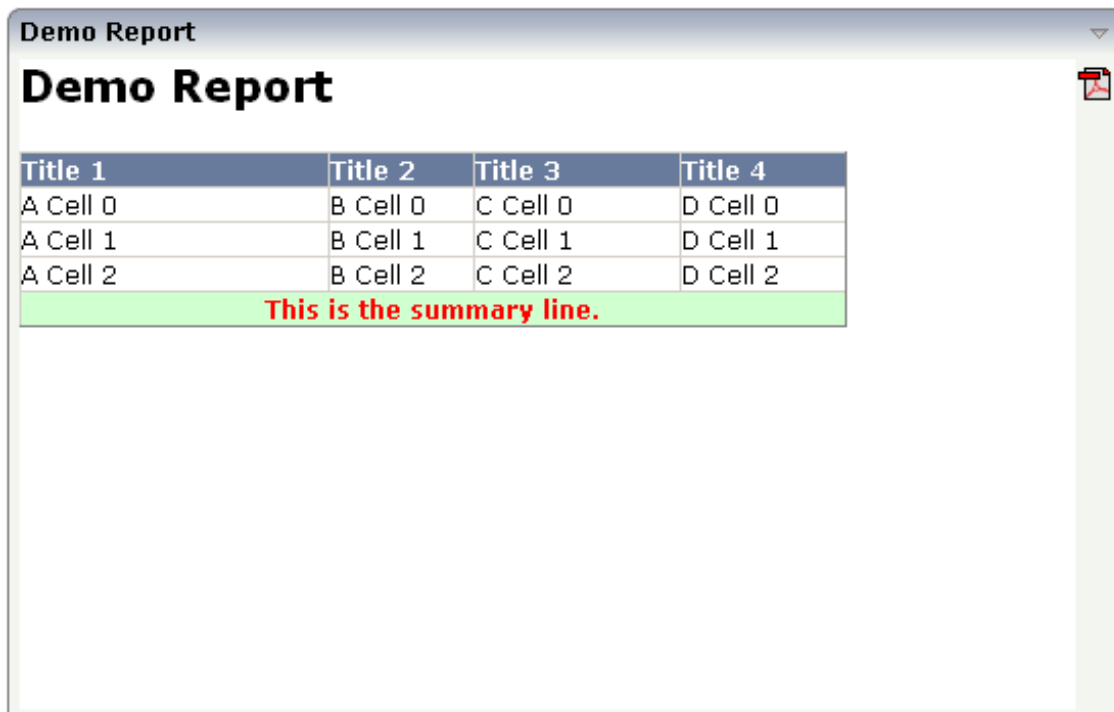
Title 1	Title 2	Title 3	Title 4
A Cell 0	B Cell 0	C Cell 0	D Cell 0
A Cell 1	B Cell 1	C Cell 1	D Cell 1
A Cell 2	B Cell 2	C Cell 2	D Cell 2

You see that grids are rendered in a simple way. By using the API of `REPORTInfo`, you can add headline and content cells. You append the grid information cell by cell, indicating new lines between.

Using Special Styles for Cell Output

The grid uses style sheet definitions that are part of the normal Application Designer style sheet files. They are taken from the `TEXTGRID` definitions inside the style sheet file.

You can override these style definitions explicitly. In the following example, the `renderGrid()` method is extended to display a summary line at the end of the grid:



The screenshot shows a window titled "Demo Report" with a sub-header "Demo Report". Below the header is a table with four columns: "Title 1", "Title 2", "Title 3", and "Title 4". The table contains three rows of data: "A Cell 0", "B Cell 0", "C Cell 0", "D Cell 0"; "A Cell 1", "B Cell 1", "C Cell 1", "D Cell 1"; and "A Cell 2", "B Cell 2", "C Cell 2", "D Cell 2". Below the table is a green summary line with the text "This is the summary line." in red.

Title 1	Title 2	Title 3	Title 4
A Cell 0	B Cell 0	C Cell 0	D Cell 0
A Cell 1	B Cell 1	C Cell 1	D Cell 1
A Cell 2	B Cell 2	C Cell 2	D Cell 2
This is the summary line.			

The code is:

```
private void renderGrid()
{
    m_report.addHeadlineCell("Title 1","150");
    m_report.addHeadlineCell("Title 2","70");
    m_report.addHeadlineCell("Title 3","100");
    m_report.addHeadlineCell("Title 4","80");
    m_report.addNewLine();
    for (int i=0; i<3; i++)
    {
        m_report.addContentCell("A Cell " + i);
    }
}
```

```

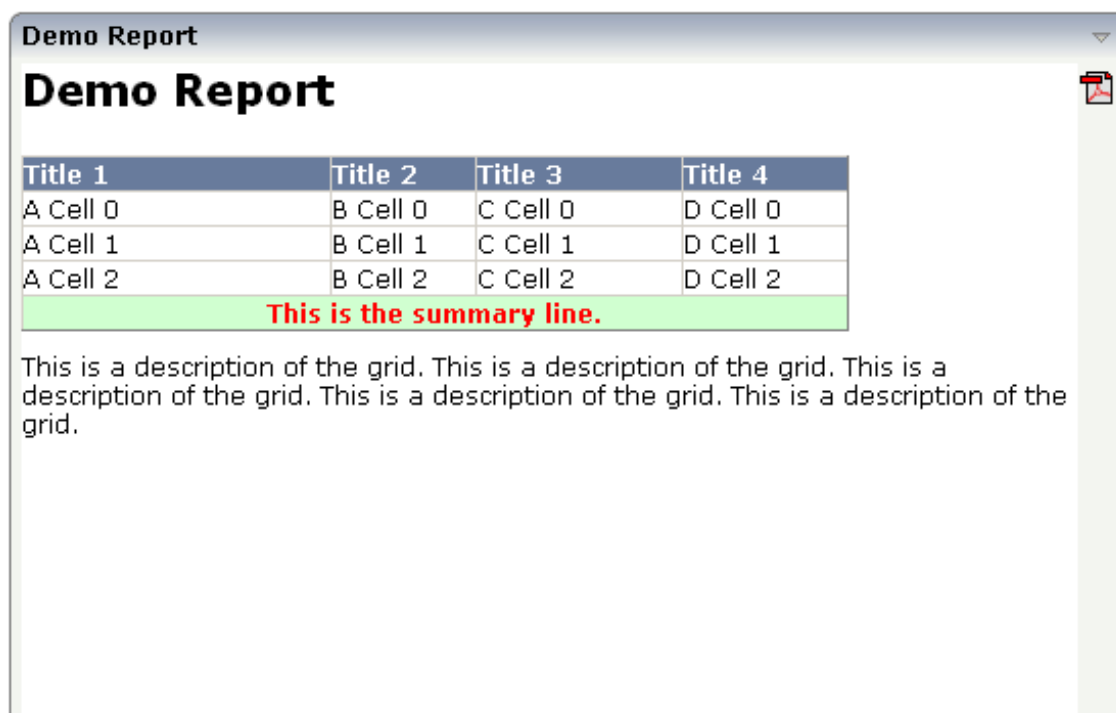
        m_report.addCell("B Cell " + i);
        m_report.addCell("C Cell " + i);
        m_report.addCell("D Cell " + i);
        m_report.addNewLine();
    }
    REPORTCellFormat sumFormat = new REPORTCellFormat();
    sumFormat.setBackgroundColor("#D0FFD0"); // light green
    sumFormat.setTextColor("#FF0000"); // red
    sumFormat.setFontWeight(REPORTCellFormat.FONTWEIGHT_BOLD);
    sumFormat.setAlign(REPORTCellFormat.ALIGN_CENTER);
    m_report.addCell("This is the summary line.", // text
                    4, // colspan
                    sumFormat); // cell format
}

```

You see that the API for adding cells offers a variant by which you can pass a columns span definition and a `REPORTCellFormat` object.

Adding Some Text

By using the `addText(...)` methods of `REPORTInfo`, you can add text to the reporting area:



The adapter code is:


```
private void renderGrid()
{
    // headline
    REPORTCellFormat hlFormat = new REPORTCellFormat();
    hlFormat.setFontSize("16pt");
    hlFormat.setFontWeight(REPORTCellFormat.FONTWEIGHT_BOLD);
    m_report.addText("Demo Report",hlFormat);
    m_report.addVerticalDistance("10");

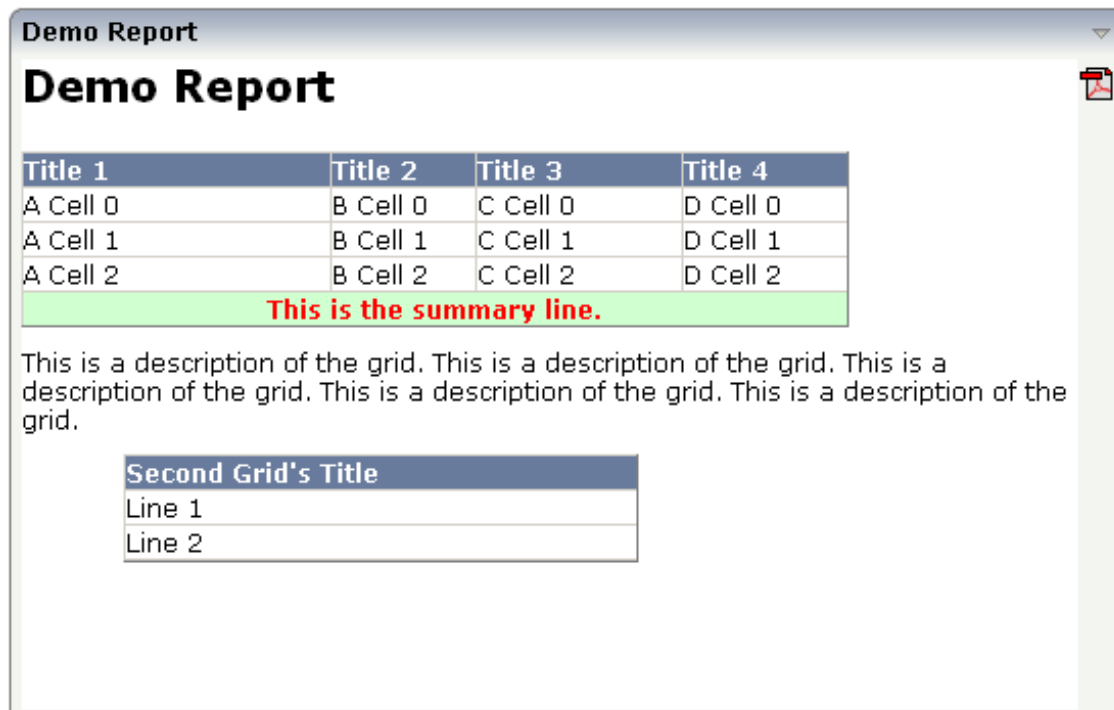
    // grid
    ...
    ...
    ... see coding above
    ...
    ...

    // description
    m_report.addVerticalDistance("10");
    m_report.addText("This is a description of the grid. " +
                    "This is a description of the grid. " +
                    "This is a description of the grid. " +
                    "This is a description of the grid. " +
                    "This is a description of the grid. ");
}
```

In the code, you see the usage of the `addText(...)` method. Text can be combined with an explicit cell format object (as done in the title) or can be output straight forward (as done in the description). In order to keep a certain distance between the texts and the grid, the method `addVerticalDistance(...)` is called.

Adding a Second Grid

You can repeat the add methods for rendering grids, texts, etc. multiple times. The formats (column width etc.) of each grid are completely independent from one another:



The code of the example was changed in the following way:

```
public void init()
{
    renderGrid();
    renderGrid2();
}

private void renderGrid()
{
    ...see coding above
}

private void renderGrid2()
{
    m_report.addVerticalDistance("10");

    m_report.setIndent("50");
    m_report.addHeadlineCell("Second Grid's Title","250");
    m_report.addNewLine();
    m_report.addCell("Line 1");
    m_report.addNewLine();
    m_report.addCell("Line 2");
    m_report.addNewLine();
}
```

By using the method `REPORTInfo.setIndent(...)`, the second grid was indented by 50 pixels.

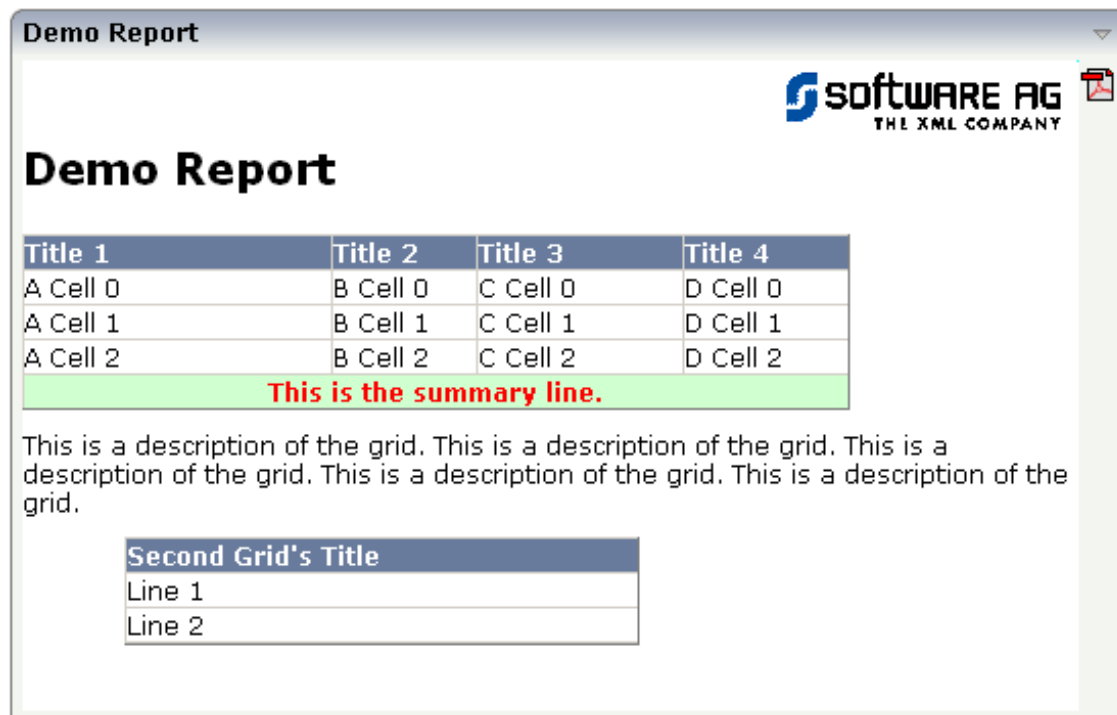
Adding an Image

Images that are part of your web application can be added by using the `addGIFGraphic(...)` or `addJPEGGraphic(...)` interface. If adding the following code to your example

```
public void init()
{
    renderLogo();
    renderGrid();
    renderGrid2();
}

private void renderLogo()
{
    m_report.addGIFGraphic("../HTMLBasedGUI/images/logo.gif","550");
}
```

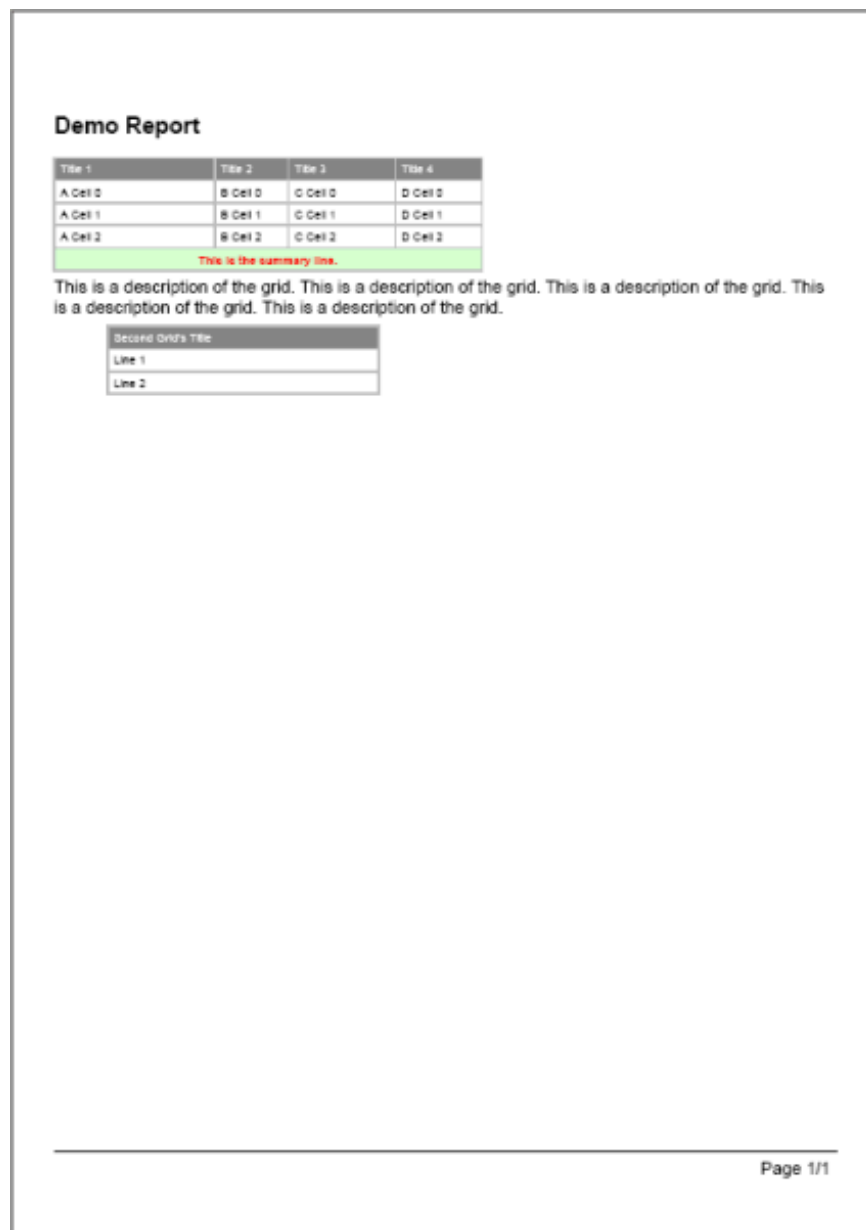
then the report will look like:



Pay attention: only add images that are accessible through your web application, e.g. that are part of your Application Designer project in which you are working. Do not add images by defining an absolute URL (*http://.....*) - this will cause problems when transferring the result to PDF.

HTML Rendering - PDF Rendering

When clicking on the PDF icon in the right top corner of the REPORT controls, the report will be rendered into PDF:



The PDF output is not a 100% match to the HTML output but a “as much as possible” match. When specifying sizes (e.g. column sizes, indent sizes) always use straight values representing pixels - as done in the examples above. There is a certain calculation factor (that can also be explicitly set by an API) between "Pixels" and PDF centimeters.

HTML tables are quite forgiving when specifying non-fitting pixel widths - there are optimisation rules that recalculate widths, e.g. if a text of a cell exceeds the size of the cell. FOP/PDF rendering is not as forgiving but will let your text overlap into the next cell. Consequence: keep an eye on your PDF output during development.

The sizing of the PDF document is done in the following way.

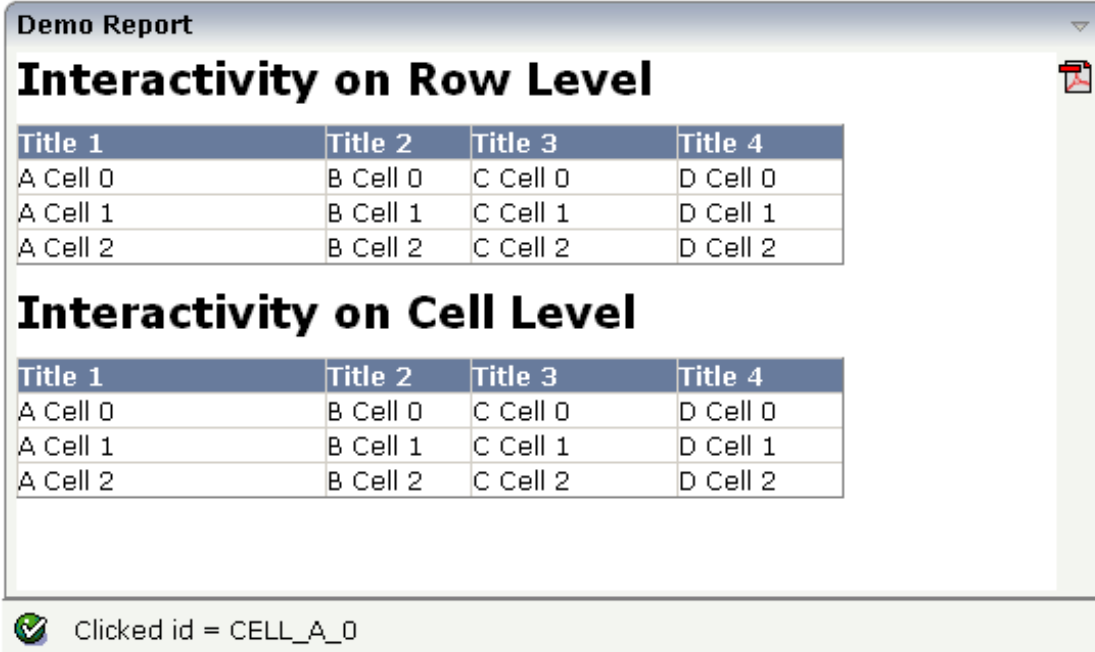
- First the system tries to render the report into an A4 paper, portrait.
- If the document exceeds the width, then the document is switched to landscape.
- If the document still exceeds the width, then the document size is increased accordingly. The page width and height will always keep the A4 relationships.

This all happens automatically - you do not have to take care of this.

Reacting on Mouse Clicks

The examples up to now only showed one way reporting. A report was just produced and output.

The REPORT controls also offer the possibility to react on mouse clicks. When a grid is clicked, certain information is passed to the adapter program that identifies the clicked object. You can either identify whole rows or single cells within rows.



Demo Report

Interactivity on Row Level

Title 1	Title 2	Title 3	Title 4
A Cell 0	B Cell 0	C Cell 0	D Cell 0
A Cell 1	B Cell 1	C Cell 1	D Cell 1
A Cell 2	B Cell 2	C Cell 2	D Cell 2

Interactivity on Cell Level

Title 1	Title 2	Title 3	Title 4
A Cell 0	B Cell 0	C Cell 0	D Cell 0
A Cell 1	B Cell 1	C Cell 1	D Cell 1
A Cell 2	B Cell 2	C Cell 2	D Cell 2

Clicked id = CELL_A_0

If clicking onto a row in the top grid, then a corresponding ID is output in the status bar. The same happens if clicking onto a cell in the bottom grid.

The adapter code is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.REPORTCellFormat;
import com.softwareag.cis.server.util.REPORTInfo;

public class Demo_Report__interactive_Adapter
    extends Adapter
{
    public class MyREPORTInfo
    extends REPORTInfo
    {
        public MyREPORTInfo(Adapter Adapter)
        {
            super(model);
        }
        public void reactOnClick(String id)
        {
            outputMessage(MT_SUCCESS, "Clicked id = " + id);
        }
    }

    // property >report<
    REPORTInfo m_report = new MyREPORTInfo(this);
    public REPORTInfo getReport() { return m_report; }

    private void renderGrid1()
    {
        //      headline
        REPORTCellFormat hlFormat = new REPORTCellFormat();
        hlFormat.setFontSize("16pt");
        hlFormat.setFontWeight(REPORTCellFormat.FONTWEIGHT_BOLD);
        m_report.addText("Interactivity on Row Level", hlFormat);
        m_report.addVerticalDistance("10");
        //      grid
        m_report.addHeadlineCell("Title 1", "150");
        m_report.addHeadlineCell("Title 2", "70");
        m_report.addHeadlineCell("Title 3", "100");
        m_report.addHeadlineCell("Title 4", "80");
        for (int i = 0; i < 3; i++)
        {
            m_report.addNewLine("ROWID_" + i);
            m_report.addContentCell("A Cell " + i);
            m_report.addContentCell("B Cell " + i);
            m_report.addContentCell("C Cell " + i);
            m_report.addContentCell("D Cell " + i);
        }
    }
}
```

```

private void renderGrid2()
{
    //      headline
    m_report.addVerticalDistance("10");
    REPORTCellFormat hlFormat = new REPORTCellFormat();
    hlFormat.setFontSize("16pt");
    hlFormat.setFontWeight(REPORTCellFormat.FONTWEIGHT_BOLD);
    m_report.addText("Interactivity on Cell Level", hlFormat);
    m_report.addVerticalDistance("10");
    //      grid
    m_report.addHeadlineCell("Title 1", "150");
    m_report.addHeadlineCell("Title 2", "70");
    m_report.addHeadlineCell("Title 3", "100");
    m_report.addHeadlineCell("Title 4", "80");
    for (int i = 0; i < 3; i++)
    {
        m_report.addNewLine();
        m_report.addContentCell("A Cell " + i, "CELL_A_" + i);
        m_report.addContentCell("B Cell " + i, "CELL_B_" + i);
        m_report.addContentCell("C Cell " + i, "CELL_C_" + i);
        m_report.addContentCell("D Cell " + i, "CELL_D_" + i);
    }
}

/** initialisation - called when creating this instance*/
public void init()
{
    renderGrid1();
    renderGrid2();
}
}

```

You see that there is an own subclass of `REPORTInfo` that is used within the adapter program. The subclass has overridden the method `reactOnClick(...)`. Through the method, it receives an ID.

The ID is set either when creating a new line or when creating a cell. You are responsible for defining the format and the semantics behind the ID.

REPORTInfo API

The previous topics showed you by example how to build reports. They did not cover the full interface of `REPORTInfo` and related classes. See the corresponding Java API documentation.

REPORT Controls versus TEXTGRID Control

The REPORT is designed to produce flexible output lists. Compared to the text grid, it has the following advantages:

- It is easier to program. The report list is generated inside the Java Code; you do not have to work both in the Layout Painter and in your Java editor.
- It is more flexible. You can span cells, and you can individually colorize them.
- It is faster. The HTML for the REPORT is created on the client side and plugged into the client's area. There is no additional dynamic rendering at the client side.

It has the following disadvantages:

- It does not offer server side scrolling, i.e. the whole report list result is always transferred to the client in one step. If it is a really large report, this may take a while, especially in WAN scenarios.
- It does not offer the same kind of interactivity but is more static. You can only click on rows. It does not offer right mouse button pop-up menus, highlighting of selections, roll over effects, switching columns, etc.

Properties

Basic			
reportprop	Name of adapter property that is referenced by the REPORT control. The adapter property must be of type "REPORTInfo". See the corresponding Java API documentation in order to get more information how to define report output for this control.	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").	Obligatory	100 150 200 250 300 250 400

	(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.		50% 100%
name	A report can be downloaded as pdf file. If you specify a value, this value will be used as file name of the pdf file. Otherwise a default file name is used.	Optional	
showpdf	<p>If set to "true" then a PDF icon is rendered in the right top corner of the control. When the user clicks on the icon then the report is automatically rendered as PDF - and the result will show up in a popup window.</p> <p>Pay attention: if setting this property to "true" then you also have to choose a special constructor when creating the REPORTInfo instance on server side, in which the instance of the model is passed as argument.</p> <p>Example:</p> <pre>public class XYZAdapter extends Adapter { REPORTInfo m_report = new REPORTInfo(this) ... }</pre>	Optional	true false
showprintversion	<p>If switched to "true" then a small print icon will appear right from the report area. The print icon opens up a modal popup from which the HTML produced inside the report can be directly sent to the printer.</p> <p>Pay attention: if specifying "true" then the adapter property holding the REPORTInfo object must create the REPORTInfo instance with passing "this" in the constructor.</p>	Optional	true false
showupload	NATPAGE layouts only: If set to "true" then a PDF icon is rendered in the right top corner of the control. When the user clicks on the icon then the report is automatically rendered as PDF and the PDF content is added to the NJX:OBJECTS cache for an upload to the Natural server. The event value-of-reportprop.onUploadPDF is triggered in the Natural application. The Natural application can	Optional	true false

	access the PDF in the NJX:OBJECTS data structure during this event.		
areastyle	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p> <p>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.</p>	Optional	<p>background-color: #FF0000</p> <p>color: #0000FF</p> <p>font-weight: bold</p>
areastyleclass	CSS style class used for rendering.	Optional	
fixlayout	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>	Optional	<p>true</p> <p>false</p>

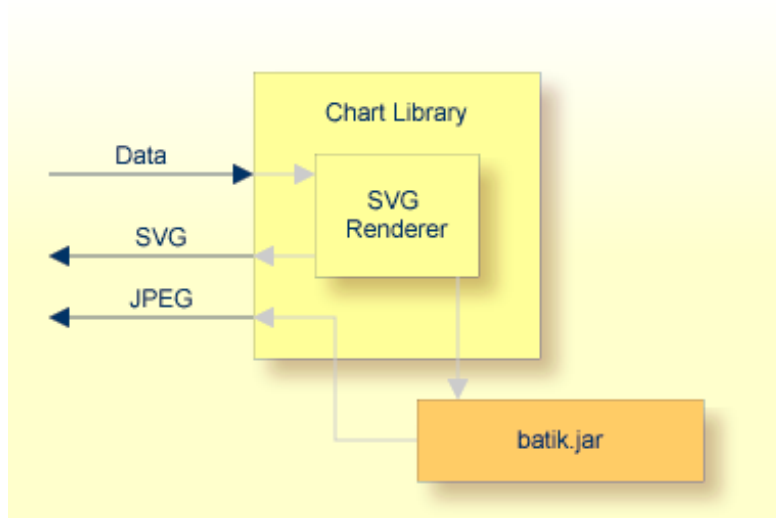
15

Creating Statistical Charts

■ Structure	74
■ Class <code>com.softwareag.cis.chart.CHARTInfo</code>	75
■ Creating a Simple Chart	75
■ Setting the Scale of the y-Axis	78

Structure

The creation of statistical charts is done by a library that is part of the Application Designer library.



The library offers an API in which you pass the data to be rendered into a chart. It interprets the data and creates an SVG (scalable vector graphics) string out of it. This SVG string can be directly passed back to the calling application.

In addition, the SVG string can be rendered into a JPEG image. Internally, the Apache *batik.jar* is used for this purpose.

When should you use which format?

- SVG is a vector format. This means that you can use it in printable documents because the quality of rendering is scalable.
- JPEG is a format for pixel images. It is easily displayable in any browser. However, it has a limited printing quality.

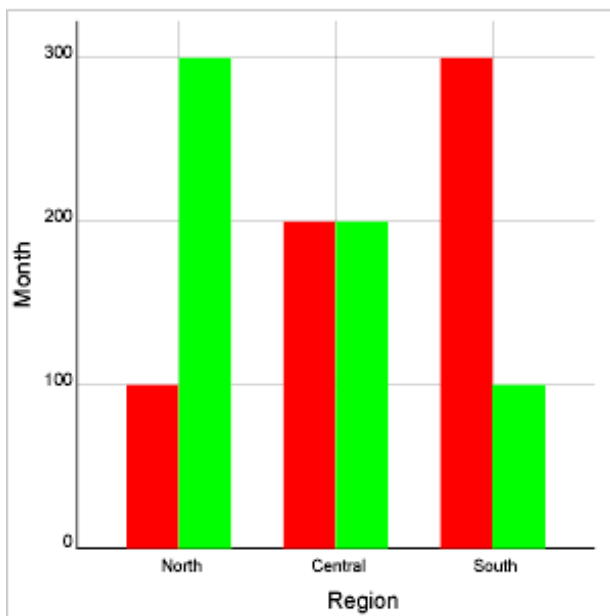
The chart library in principle is a self-containing library, packaged into the *cis.jar*. The result can be flexibly used inside Application Designer.

Class `com.softwareag.cis.chart.CHARTInfo`

The class `CHARTInfo` is the central class for creating charts. See the Java API documentation. The information below will only provide an example of how to use this library.

Creating a Simple Chart

The following chart will be created:



Let us take a look at the Java program that creates the chart:

```
private void renderChart()
{
    CHARTInfo chart = buildChart();
    String svg = chart.getSVGBarChart();
}
private CHARTInfo buildChart()
{
    // CHARTInfo chart = new CHARTInfo("300","300"); // width/height
    // set descriptions
    chart.setXAxisDescription("Region","#000000");
    chart.setYAxisDescription("Month","#000000");
    // Define x-axis
    chart.addXAxisScale("North");
    chart.addXAxisScale("Central");
```

```

chart.addXAxisScale("South");
// Define "value lines"
chart.addDataDescription("January", "#FF0000");
chart.addDataDescription("February", "#00FF00");
// Define values
chart.addData("North", "January", 100);
chart.addData("Central", "January", 200);
chart.addData("South", "January", 300);
chart.addData("North", "February", 300);
chart.addData("Central", "February", 200);
chart.addData("South", "February", 100);
return chart;
}

```

The program is split into two methods: one method (`buildChart`) fills a `CHARTInfo` object with data. The other method (`renderChart`) takes the result and triggers the output of the SVG information.

First have a look at the `buildChart()` method:

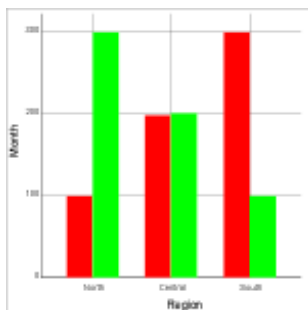
- A `CHARTInfo` object is initialized.
- The names of x- and y-axis are defined.
- The scale of the x-axis is defined.
- Value lines are defined - each value line represents one set of values that forms one line.
- Data is passed into the `CHARTInfo` object.

At the end, the chart information is logically filled.

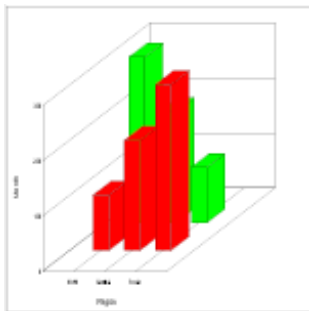
The rendering is triggered at the point of time when accessing the `CHARTInfo`'s `getSVG*()` or `getJPEG*()` methods. This is done in the example's `renderChart()` method. For each type of graphic, you have a different method.

Examples:

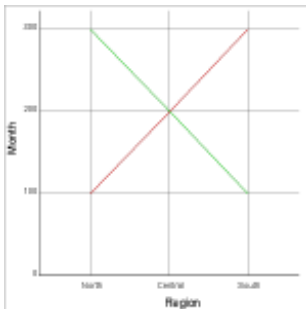
■ `getSVGBarChart()`



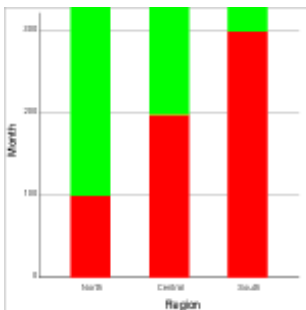
■ getSVG3DBarChart()



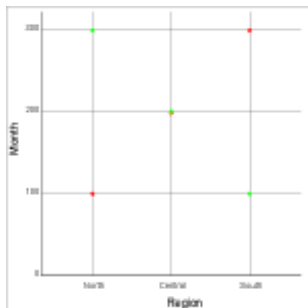
■ getSVGLineChart()



■ getSVGBarStackedChart()



■ getSVGPointChart()



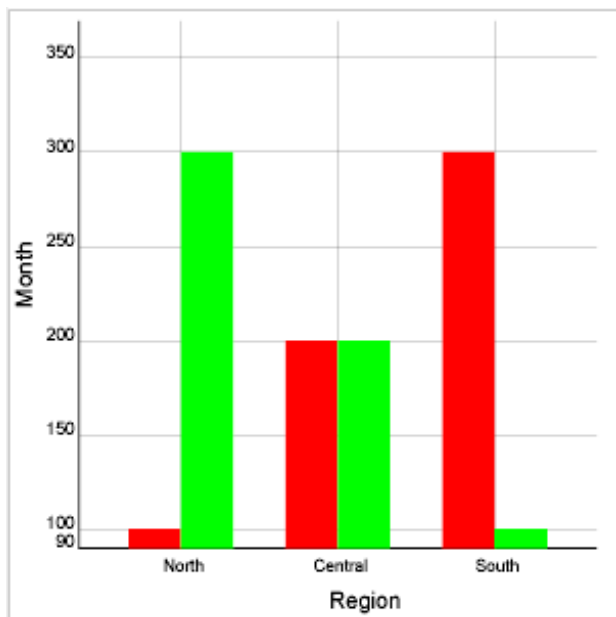
The SVG methods return an SVG string. There are corresponding other methods that return a JPEG file.

Setting the Scale of the y-Axis

If the scale of the y-axis is not specified explicitly, the scale will be calculated automatically:

- In case of only positive data values, the bottom of the scale will be "0". The top will be the maximum data value that is found. Between "0" and the maximum value, some interim scale figures will be added that are clean decimal figures.
- In case of both positive and negative values, the bottom of the scale will be the minimum value.

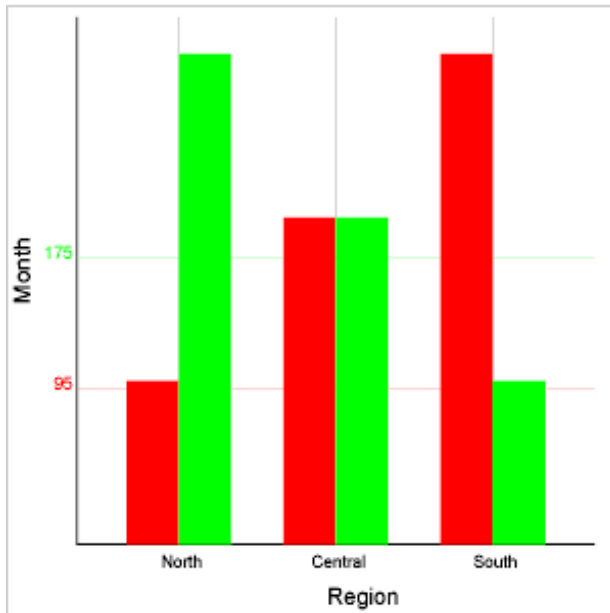
You can also set the scale on your own.



To do so, use the following method:


```
chart.setYAxisDimension(90,350);
```

The system will still try to find optimal interim scale values. However, you also can set the y-scale completely on your own:



The code for doing so is:

```
chart.addYAxisScale(95,"#FF0000");  
chart.addYAxisScale(175,"#00FF00");
```


16

Embedding Statistical Charts into Reports

■ Creating an SVG Graphic and Embedding it into a Report	82
■ Creating a JPEG Graphic and Embedding it into a Report	83
■ Pay Attention when Sizing your Graphic	84

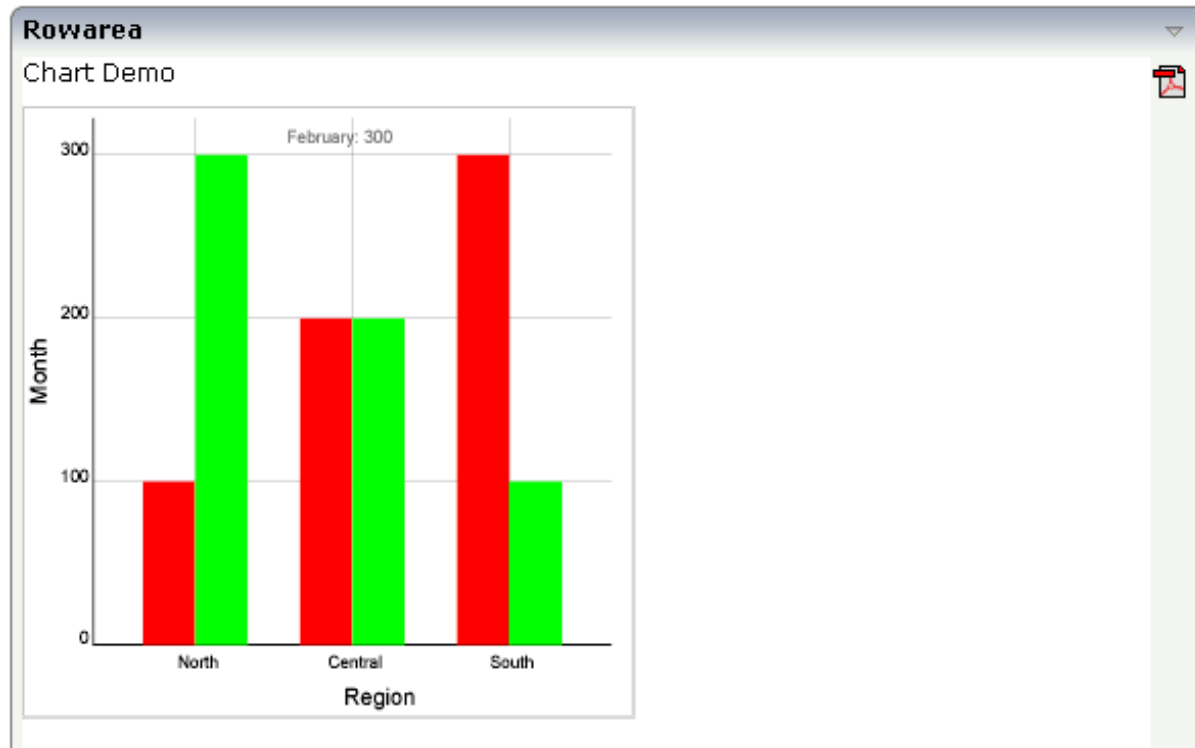
The chart library is not bound to the reporting functions, but can of course be used there in a simple way. You can produce charts and directly embed them into your report. The `REPORTInfo` class hides the internal complexity that is involved in doing this.

Creating an SVG Graphic and Embedding it into a Report

Have a look at the following code:

```
private void renderChart()
{
    // headline in front of chart
    m_report.addText("Chart Demo");
    CHARTInfo chart = buildChart();
    m_report.addVerticalDistance("10");
    String svg = chart.getSVGBarChart();
    m_report.addSVGsource(svg,"300", // width
        "300", // height
        "0"); // indent
}
private CHARTInfo buildChart()
{
    // CHARTInfo chart = new CHARTInfo("300","300"); // width/height
    // set descriptions
    chart.setXAxisDescription("Region","#000000");
    chart.setYAxisDescription("Month","#000000");
    // Define x-axis
    chart.addXAxisScale("North");
    chart.addXAxisScale("Central");
    chart.addXAxisScale("South");
    // Define "value lines"
    chart.addDataDescription("January","#FF0000");
    chart.addDataDescription("February","#00FF00");
    // Define values
    chart.addData("North","January",100);
    chart.addData("Central","January",200);
    chart.addData("South","January",300);
    chart.addData("North","February",300);
    chart.addData("Central","February",200);
    chart.addData("South","February",100);
    return chart;
}
```

It creates an SVG graphic by using `CHARTInfo` and directly places the SVG graphic into a report. The output looks as follows:



Creating a JPEG Graphic and Embedding it into a Report

Nearly the same code is required for creating the graphic as an JPEG file:

```
private void renderChart()
{
    ...
    String svg = chart.getSVGBarChart();
    byte[] jpegBytes = chart.transformChartToGraphic(svg, CHARTInfo.CREATEJPEGIMAGE);
    m_report.addJPEGsource(jpegBytes);
    ...
}

private CHARTInfo buildChart()
{
    ...
    ...
    ...
}
```

By using the `transformChartToGraphic()` method, you can translate generated SVG into various file formats (JPEG, PNG, TIFF).

Pay Attention when Sizing your Graphic

You may already have recognized that in the previous examples the `CHARTInfo` object was initialised in the following way:

```
CHARTInfo chart = new CHARTInfo("300","300");
```

On the other hand, when placing the generated SVG into the report output, again the size was passed:

```
m_report.addSVGsource(svg,"300", // width  
                      "300", // height  
                      "0"); // indent
```

Make sure that both sizes are the same.

When using SVG for reporting, you do not pass sizes like "4 cm" but you pass pixel values. Pixel values are transferred into metric sizes within a calculation factor. You can change the calculation factor by using the method `REPORTInfo.setFactorForTransformPixel2PDF(...)`.

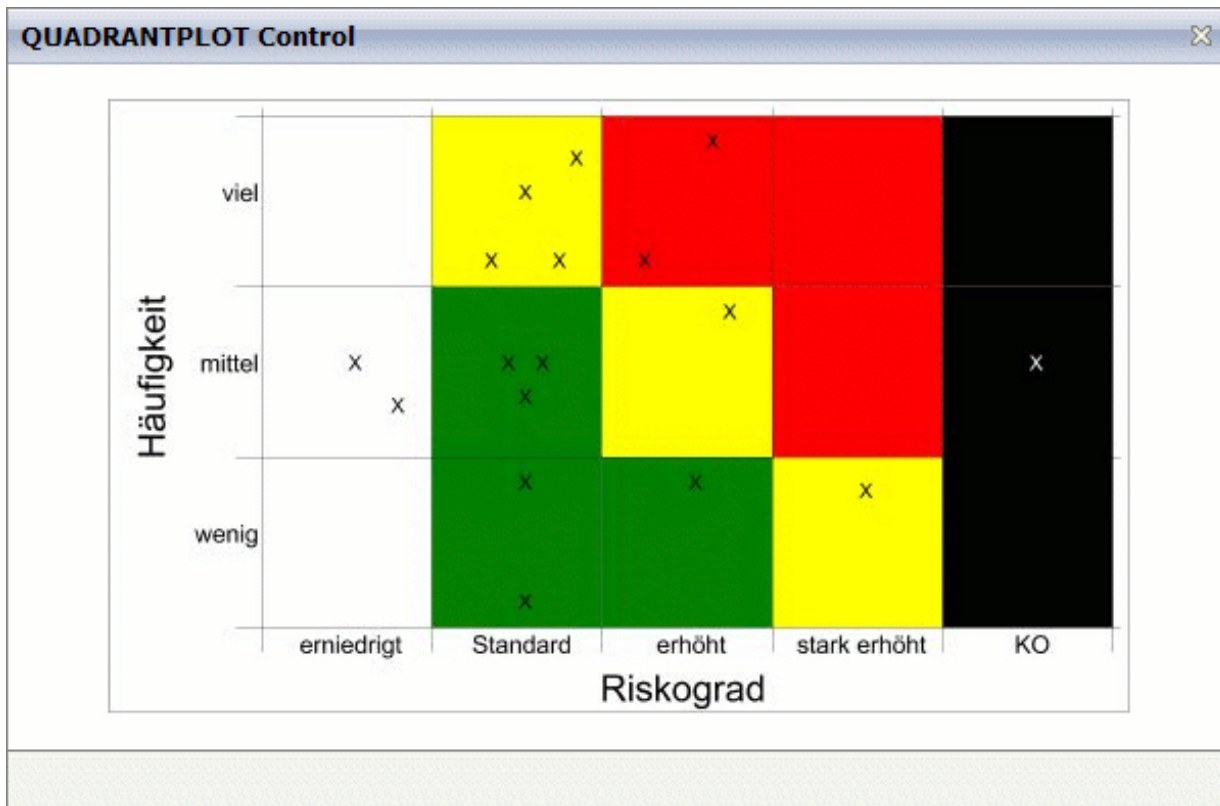
17

Using the Special Chart Control QUADRANTPLOT

■ Simple Example	86
■ Properties	89

The QUADRANTPLOT control represents an extension to the normal SVG chart. It allows you to divide the displayed chart into quadrants and to react on clicks.

Simple Example



The XML layout definition is:

```
<pagebody takefullheight="true">
  <itr takefullwidth="true" valign="middle">
    <quadrantplot quadrantplotprop="drawArea">
    </quadrantplot>
  </itr>
</pagebody>
```

The Java code of the adapter is:


```

public class QuadrantPlotAdapter
    extends Adapter
    implements IQUADRANTPLOTListener
    // Interface for the click event of the QUADRANTPLOTInfo.

{
    QUADRANTPLOTInfo m_drawArea = new QUADRANTPLOTInfo(this);
    public QUADRANTPLOTInfo getDrawArea() { return m_drawArea; }
    public void setDrawArea(QUADRANTPLOTInfo value) { m_drawArea = value; }

    /** Interface IQUADRANTPLOTListener
     * Method is call after a click in a quadrant with an ID.<br>
     * The ID is passed as argument. */
    public void reactOnClick(String id)
    {
        outputMessage (MT_SUCCESS,"Quadrant "+ id + " was clicked!");
    }

    public void initData()
    {
        // set x axis quadrants description
        String[] xAxisValues = new String[]
        {
            "decreased", "standard", "increased", "highly increased", "none"
        };

        // set y axis quadrants description
        String[] yAxisValues = new String[]
        {
            "low", "mid", "high"
        };

        m_drawArea.clearDrawAreaData();
        m_drawArea.rotateXAxisScaleText(-0,0);
        m_drawArea.setLeftYAxisDistance(400);
        m_drawArea.printFrameAround(true);

        // set x axis description
        m_drawArea.setXAxisDescription("Risk","black", 18, 250);
        m_drawArea.setYAxisDescription("Occurrences","black", 18, 250);

        m_drawArea.setXAxis(xAxisValues);
        m_drawArea.setYAxis(yAxisValues);

        // added quadrants
        m_drawArea.addQuadrantInfo(0, 0, "white", "area_0_0");
        m_drawArea.addQuadrantInfo(0, 1, "white", "area_0_1");
        m_drawArea.addQuadrantInfo(0, 2, "white", "area_0_2");

        m_drawArea.addQuadrantInfo(1, 0, "green", "area_1_0");
        m_drawArea.addQuadrantInfo(1, 1, "green", "area_1_1");
        m_drawArea.addQuadrantInfo(1, 2, "yellow", "area_1_2");
    }
}

```

```
m_drawArea.addQuadrantInfo(2, 0, "green", "area_2_0");
m_drawArea.addQuadrantInfo(2, 1, "yellow", "area_2_1");
m_drawArea.addQuadrantInfo(2, 2, "red", "area_2_2");

m_drawArea.addQuadrantInfo(3, 0, "yellow", "area_3_0");
m_drawArea.addQuadrantInfo(3, 1, "red", "area_3_1");
m_drawArea.addQuadrantInfo(3, 2, "red", "area_3_2");

m_drawArea.addQuadrantInfo(4, 0, "black", "area_4_0");
m_drawArea.addQuadrantInfo(4, 1, "black", "area_4_1");
m_drawArea.addQuadrantInfo(4, 2, "black", "area_4_2");

// added values
m_drawArea.addPlotData(0.5, 1.5, "X", 10, "black");
m_drawArea.addPlotData(0.75, 1.25, "X", 10, "black");

m_drawArea.addPlotData(1.5, 0.1, "X", 10, "black");
m_drawArea.addPlotData(1.5, 0.8, "X", 10, "black");

m_drawArea.addPlotData(1.5, 1.3, "X", 10, "black");
m_drawArea.addPlotData(1.4, 1.5, "X", 10, "black");
m_drawArea.addPlotData(1.6, 1.5, "X", 10, "black");

m_drawArea.addPlotData(1.3, 2.1, "X", 10, "black");
m_drawArea.addPlotData(1.7, 2.1, "X", 10, "black");
m_drawArea.addPlotData(1.5, 2.5, "X", 10, "black");
m_drawArea.addPlotData(1.8, 2.7, "X", 10, "black");

m_drawArea.addPlotData(2.5, 0.8, "X", 10, "black");

m_drawArea.addPlotData(2.7, 1.8, "X", 10, "black");

m_drawArea.addPlotData(2.2, 2.1, "X", 10, "black");
m_drawArea.addPlotData(2.6, 2.8, "X", 10, "black");

m_drawArea.addPlotData(3.5, 0.75, "X", 10, "black");

m_drawArea.addPlotData(4.5, 1.5, "X", 10, "white");

// get the svg plot
m_drawArea.setSVGOutputWidth(300);
}

/** initialisation - called when creating this instance*/
public void init()
{
    initData();
}
}
```

Properties

Basic			
quadrantplotprop	<p>Name of the adapter property that represents the control on server side.</p> <p>Return type must be "QUADRANTPLOTInfo".</p> <p>Pay attention: The QUADRANTPLOTInfo Constructor needs a valid Adapter i.e.</p> <p>QUADRANTPLOTInfo m_qpi = new QUADRANTPLOTInfo(this) ;</p>	Obligatory	
height	<p>Height of the control.</p> <p>There are three possibilities to define the height:</p> <p>(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.</p> <p>(B) Pixel sizing: just input a number value (e.g. "20").</p> <p>(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.</p>	Optional	100 150 200 250 300 250 400 50% 100%

18

Creating Simple Charts Quickly Using the PIVOT Control

■ Simple Example	92
■ Properties	95

The PIVOT control offers an easy way to create interactive pivot tables in a short period of time. All that is required is passing an array of data to the `PivotChartInfo` object which is the server-side representation of the PIVOT control.

Simple Example

Pivot Example

Calculation ☒ Summation ☐ Maximum
☐ Average ☐ Minimum
☐ Count

Country	Product	Region	Revenue	Pieces
D	Laptop	WHOLE	78.0	25.0
	Total Laptop		78.0	25.0
	PC	WHOLE	115.0	43.0
	Total PC		115.0	43.0
Total D			193.0	68.0
US	Laptop	CA	100.0	10.0
		MI	120.0	10.0
		NY	110.0	20.0
		OR	130.0	40.0
	Total Laptop		460.0	80.0
	PC	CA	100.0	33.0
		MI	120.0	31.0
		NY	110.0	45.0
		OR	130.0	32.0
	Total PC		460.0	141.0
Total US			920.0	221.0
Total			1113.0	289.0

The XML layout definition is:

```
<pagebody>
  <rowarea name="Simple Pivot Chart Demo">
    <pivot pivotprop="pivot" height="390" showpdf="true">
      </pivot>
    </rowarea>
  </pagebody>
```

The Java code of the adapter is:

```

PIVOTInfo m_pivot = new PIVOTInfo(this);
public PIVOTInfo getPivot() { return m_pivot; }
public void setPivot(PIVOTInfo value) { m_pivot = value; }

/** initialization - called when creating this instance */
public void init()
{
    PIVOTDataArray pda = new PIVOTDataArray
    (
        new String[] // Column Headers
        {
            "Country","Product","Region","Revenue","Pieces"
        },
        new String[][] // Column Content
        {
            { "D","Laptop","WHOLE","100","10" },
            { "D","PC","WHOLE","110","20" },
            { "US", "Laptop","CA","120","10" },
            { "US", "Laptop","MI","130","40" },
            { "US", "Laptop","NY","100","33" },
            { "US", "Laptop","OR","110","45" },
            { "US","PC","CA","120","31" },
            { "US","PC","MI","130","32" },
            { "US","PC","NY","78","25" },
            { "US","PC","OR","115","43" }
        }
    );

    pda.setGroupColumns(new int[] {0,1,2});
    pda.setDataColumns(new int[] {3, 4});

    m_pivot.setComputedValueType(m_pivot.COMPUTE_SUM);
    m_pivot.setPivotData(pda);
}

```

The PIVOT control is represented by an `PIVOTInfo` object on the adapter side. When you look at the constructor, you will see that the `PIVOTInfo` constructor needs a valid adapter, that is `new PIVOTInfo(this)`.

To generate a simple pivot table, you just need to fill a `PIVOTDataArray` object with two string arrays. The first string array contains the column headers (that is "Country", "Product", "Region", "Revenue" and "Pieces"). The second string array holds the contents of the rows.

In this simple example, the third row contains the following data: "US", "Laptop", "CA", "120" and "10".

The server needs to know which columns contain data that are relevant for sorting and which columns only contain descriptive information that can be used for grouping the columns.

In this example, the first three columns are the so-called "group columns" which only contain descriptive information. You define these three group columns as follows:

```
pda.setGroupColumns(new int[]{0,1,2});
```

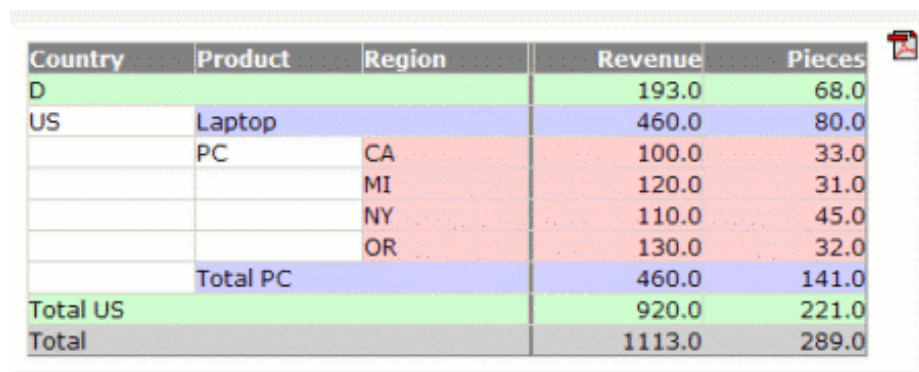
You define the columns which contain the data in a similar way:

```
pda.setDataColumns(new int[] {3, 4});
```

There are different ways of sorting or grouping the PIVOT table. The default value is `COMPUTE_SUM`. You can change this value dynamically using the `setComputedValueType()` method. Valid values for this method are:

Constant	Value
COMPUTE_SUM	0
COMPUTE_AVERAGE	1
COMPUTE_MAX	2
COMPUTE_MIN	3
COMPUTE_COUNT	4

You can change the detail of the corresponding level of each group by simply clicking in the pivot table:



Country	Product	Region	Revenue	Pieces
D			193.0	68.0
US	Laptop		460.0	80.0
	PC	CA	100.0	33.0
		MI	120.0	31.0
		NY	110.0	45.0
		OR	130.0	32.0
	Total PC		460.0	141.0
Total US			920.0	221.0
Total			1113.0	289.0

The first row now just shows the sum of data group "D"; and within the data group "US", the "Laptop" group also just shows the sum.

Properties

Basic			
pivotprop	Name of adapter property that represents the pivot report on server side. Must be of type "PIVOTInfo".	Obligatory	
height	<p>Height of the control.</p> <p>There are three possibilities to define the height:</p> <p>(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.</p> <p>(B) Pixel sizing: just input a number value (e.g. "20").</p> <p>(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.</p>	Obligatory	100 150 200 250 300 250 400 50% 100%
showpdf	<p>If set to "true" then a PDF icon is rendered in the right top corner of the control. When the user clicks on the icon then the report is automatically rendered as PDF - and the result will show up in a popup window.</p> <p>Pay attention: if setting this property to "true" then you also have to choose a special constructor when creating the REPORTInfo instance on server side, in which the instance of the model is passed as argument.</p> <p>Example:</p> <pre>public class XYZAdapter extends Adapter { REPORTInfo m_report = new REPORTInfo(this) ... }</pre>	Optional	true false

areastyle	<p>CSS style definition that is directly passed into this control.</p> <p>With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are:</p> <p>border: 1px solid #FF0000</p> <p>background-color: #808080</p> <p>You can combine expressions by appending and separating them with a semicolon.</p> <p>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.</p>	Optional	<p>background-color: #FF0000</p> <p>color: #0000FF</p> <p>font-weight: bold</p>
-----------	--	----------	---

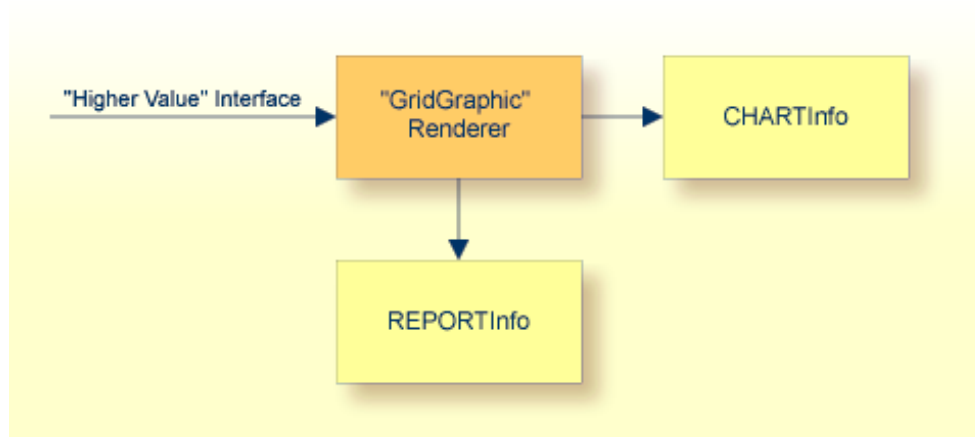
19

Best Practice Hints

The reporting functions you got to know in this part enable you to do reporting output including interactive grids, texts and graphics. Before directly using them inside your programs, you should think about setting up a certain abstraction layer on top of the functions that has a higher degree of semantic relationship with your application.

What do we mean? Let us explain by example:

Imagine you often have the situation of showing grids of data together with a certain graphic. In this case, you could think about writing a “report renderer” that receives an array of values and that produces grid and graphics in one step. It internally uses the APIs of Application Designer reporting:



Maybe you even can embed some data collection functions into your “report renderer” functions that pick certain data from a data source and immediately transfer this data into a reporting output.

In other words: the Application Designer functions for reporting are technical rendering functions. Not less - and not more!

III

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:

AUTOCOMPLETE

TIMER

Extended Hot Key Management

20

AUTOCOMPLETE

■ General Information	102
■ Example	103
■ Data Sources for Populating the Values	103
■ Properties	106

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 `autocompleteid` property of the FIELD control.

```
<autocomplete id="myautocompleteid" ...></autocomplete>
<field valueprop="myinputcontrol" autocompleteid=" 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 `autocomplete_displayname` and `autocomplete_displayref` 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" autocomplete="mycity"
    autocomplete="code" autocomplete="displayname"></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 `autocomplete` and `autocomplete="displayname"` 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: <code>./autocomplete/myfile</code> , <code>http://myremotedatasource...</code>	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	

21

TIMER

▪ Example	110
▪ Properties	112

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.

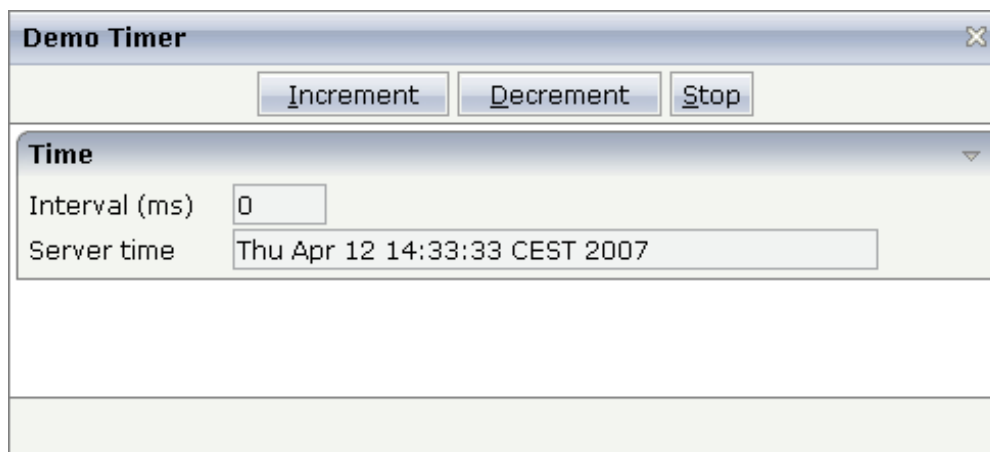
The timer tag is accessible as a valid subnode inside the page tag.

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 screenshot shows a web application window titled "Demo Timer". At the top, there are three buttons: "Increment", "Decrement", and "Stop". Below these buttons is a section titled "Time" which contains two input fields. The first field is labeled "Interval (ms)" and has the value "0". The second field is labeled "Server time" and displays the text "Thu Apr 12 14:33:33 CEST 2007".

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	

22 Extended Hot Key Management

■ Direct Hot Key Definitions with Certain Controls	114
■ Hot Key Definitions for Certain Controls	114

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.

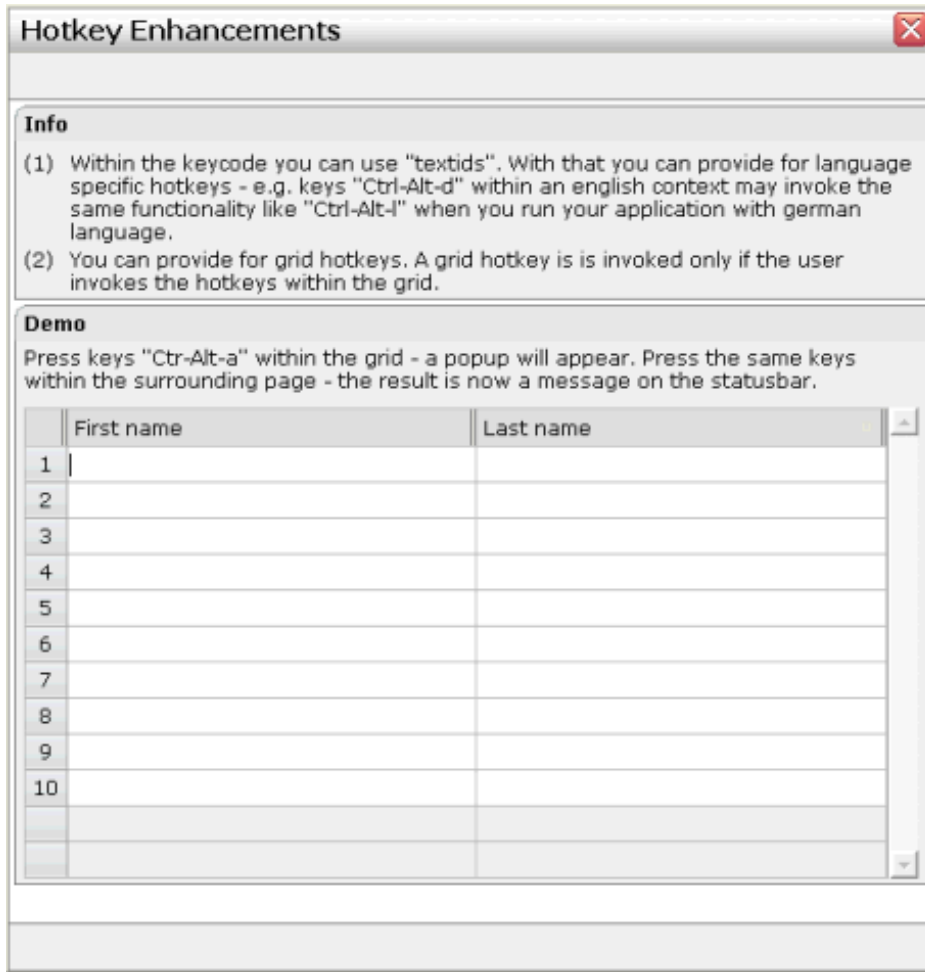
Why use CTRL+ALT as a default way to trigger the hot keys? This is because most of the simple ALT keys are already occupied by the browser.

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.

IV

Binding between Page and Adapter

One of the basic concepts of the Application Designer environment is to provide a simple mechanism for transferring data between the page which runs inside the browser and the adapter object which runs inside Application Designer. The page renders content, while the adapter provides content.

Between the page and the adapter, there is a binding which is defined during development time:

- A page is bound to an adapter class by the PAGE tag.
- Controls are bound to properties and methods of the adapter class.

At runtime, this binding definition is used inside Application Designer for accessing the adapter objects to pull and push data.

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

Phases of Adapter Processing

Class Binding

Types of Property Binding

Java Bean Property Binding

Dynamic Access Property Binding

XML Property Binding

Getting Information about Access Paths

Exception Management Inside an Adapter Object

Additional Interfaces

23

Phases of Adapter Processing

■ SET/INVOKE/GET Phase - The Default Phases	120
■ INIT Phase when Adapter is Constructed	121
■ DESTROY Phase when Adapter is Deregistered	122

SET/INVOKE/GET Phase - The Default Phases

An adapter object is the logical representation of a page. The page runs inside the GUI client, the adapter runs inside the Java server.

The user changes information on the page (e.g. inputs some values into field controls) and operates some functions (e.g. chooses a button). Every time a function is invoked, a request is initiated from the client. The request contains all data that was changed on the client, and it contains the command (e.g. the method to be called; sometimes there is no explicit command but the request just is a “data transfer request”, e.g. when having defined `FLUSH="server"` with a `FIELD` control).

The request is processed in three phases:

- (activate)
- SET phase
- INVOKE phase
- GET phase
- (passivate)

During the SET phase, Application Designer passes all changed property values into the property representations of the server side adapter object. In the following INVOKE phase, the method is called that is associated with the request. In the GET phase, Application Designer checks all property values if they have changed. If a change happened (i.e. during the INVOKE phase, some property values were changed), then the changes are communicated back as response to the client.

The SET and GET phases have dedicated methods which are called inside the adapter in order to signalize the start and end of the phases:

■ SET phase:

```
reactOnDataTransferStart();
set...();
set...();
set...();
reactOnDataTransferEnd();
```

■ GET phase:

```

reactOnDataCollectionStart();
get...();
get...();
get...();
reactOnDataCollectionEnd();

```

You can use the methods for diverse purposes:

- `reactOnDataTransferStart()`
You may want to initialize certain internal data that needs to be initialized for each request processing.
- `reactOnDataTransferEnd()`
You may want to check which properties actually have changed and which application checks have to be invoked.
- `reactOnDataCollectionStart()`
You may want to build up some interim objects for complex data structures that allow a faster response for the following get calls.
- `reactOnDataCollectionEnd()`
You may want to set certain data to initial values - in case they were changed during request processing.

INIT Phase when Adapter is Constructed

The INIT phase is processed only once per adapter instance - at the point of time when it is constructed.

The INIT phase is internally processed in two steps:

- Creation of the adapter object via the `new` operator (without any parameters).
- `init()`

Application Designer first creates an instance of an adapter object and then calls the `init()` method of the object.



Important: Many functions inside your adapter class (extended from Application Designer's Adapter class) are only available after having constructed the object. Application Designer first creates the instance of the object and then internally registers the object inside its internal data structures (session management, etc.). The `init()` method is called after the internal registration. All functions that require the adapter object to be correctly registered will fail when being called inside the constructor, but will not fail if called in the `init()` method.

Best practice: use the `init()` method as constructor for an adapter object.

DESTROY Phase when Adapter is Deregistered

The DESTROY phase is processed only once per adapter instance - when Application Designer has internally deregistered the adapter instance:

The DESTROY phase consists of one method that is called inside the adapter:

- `destroy()`

Application Designer's session management deregisters all adapters when a subsession or session is closed. All associated adapters are internally deregistered so that they are available for garbage collection. In addition, each instance receives a destroy signal so that it can internally pass back resources that it used.

24

Class Binding

■ Direct Class Binding	124
■ Generic Class Binding	125

For each page, there must be one adapter class. The name of the adapter class is given inside the PAGE tag of a page by the corresponding `model` property. Several page definitions can point to one adapter class, for example, when you have several page variants to display the same logical content.

The Application Designer runtime can create an adapter instance of a page in two ways:

1. Direct Class Binding

The Application Designer runtime directly tries to create an adapter instance from the name in the `model` property of the PAGE tag of the page.

2. Generic Class Binding

If the direct class binding does not succeed, the Application Designer runtime tries to create a generic adapter instance that has to be made accessible by the application.

Option 1 is the typical one. Option 2 is a solution if you require an adapter to be used very generically inside your framework.

Direct Class Binding

The following page definition forces the Application Designer runtime to look for a `Test1Adapter` class inside the package `com.softwareag.cis.test`:

```
<page model="com.softwareag.cis.test.Test1Adapter">
  ...
  ...
  ...
</page>
```

The class itself is derived typically from the class `com.softwareag.cis.server.Adapter`:

```
package com.softwareag.cis.test;

import com.softwareag.cis.server.Adapter;

public class Test1Adapter extends Adapter
{
    // -----
    // constructor - without parameters
    // -----

    public Test1Adapter()
    {
    }

    // -----
    // public access
    // -----
}
```

```
// -----

/** The init message is called when an object is created and all
 * runtime aspects are correctly set inside the adapter. */
public void init()
{
    ...
    ...
}
}
```

The default constructor is required (without any parameters).

Generic Class Binding

If the runtime does not find the class, it tries to find a generic one. The name of the generic class is created in the following way:

- The package name is taken from the `model` property of the PAGE tag of your page definition.
- The class name is "GenericAdapter".

Example: if you bind a page to the class `com.softwareag.cis.test.Test2Adapter` and the runtime system cannot locate the class `Test2Adapter`, the system tries to load the class `GenericAdapter` in the same package as the class that could not be found:

```
<page model="com.softwareag.cis.test.Test2Adapter">
    ...
    ...
    ...
</page>
```

The generic adapter is just a normal adapter which typically supports the dynamic binding of properties (see below).

```
package com.softwareag.cis.test;

import com.softwareag.cis.server.Adapter;

public class GenericAdapter extends Adapter
{
    /** */
    public void init()
    {
        System.out.println("My original class name is " + this.m_modelName);
    }
}
```

Each adapter object can access the `m_modelName` member. This member is set after the initialisation of the object. It holds the original adapter name to which the page refers.

25

Types of Property Binding

There are different types of binding techniques that are provided:

- **Java Bean binding** - the adapter provides set/get methods.
- **Dynamic access binding** - the adapter provides the implementation of a generic interface to access its data.
- **XML access binding** - the property values are kept within XML files, together with the page layout.

26

Java Bean Property Binding

■ Class Binding	130
■ Method Binding	130
■ Property Binding	131
■ Access Path Restrictions	134

Class Binding

The page binding is defined in the PAGE tag of your page definition. The PAGE tag points to a class supporting the interface `com.softwareag.cis.server.IModel`. There is a class `com.softwareag.cis.server.Adapter` which implements this interface - which should be used to build adapter classes as subclasses of `Adapter`.

Example:

```
<page model="com.softwareag.cis.demo.DemoAdapter" ...>
  ...
  ...
  ...
</page>
```

The above definition points to a class which looks as follows:

```
package com.softwareag.cis.demo;

import com.softwareag.cis.server.*;

public class DemoAdapter
    extends Adapter
{
    // constructor - either no constructor or a constructor
    // without any parameters
    public DemoAdapter()
    {
    }
    ...
    ...
}
```

Note that the adapter class has at least a default constructor (without any parameters).

Method Binding

Controls triggering a method inside the adapter are bound to a method name of the adapter. The method implementation itself must be a method without any parameters.

Example:

```
<button name="Save" method="doSave" ...>
</button>
```

The above button definition points to a method inside the adapter class which looks as follows:

```
public void doSave()
{
    ...
    ...
}
```

Property Binding

Controls presenting or manipulating data of the adapter are bound to properties of the adapter. There is a flexible concept available that makes it possible for you to use the following:

- Simple Properties which are Provided Directly by the Adapter
- Simple Properties which are Provided by Embedded Objects of the Adapter
- Array Properties which are Provided Directly by the Adapter
- Array Properties which are Provided by Embedded Objects of the Adapter

Simple Properties which are Provided Directly by the Adapter

This is the easiest way of binding: the property name which you specify in the definition of the control is provided directly by the adapter object - by a corresponding set and get method. It depends on the control whether you have to provide both set and get methods or just one of them.

Following the Java Bean conventions, the first character of the property name is written as a capital letter inside the corresponding set or get method.

The get method must return a value which is either a simple data type or a “simple” object. A list of supported return values is shown in *Appendix C - Data Types to be Used by Adapter Properties*.

The set method must offer one parameter to update its value at runtime. The parameter type must either be a simple data type or one of the classes that are listed in appendix C.

Example:

```
<field valueprop="name" ...></field>
<field valueprop="age" ...></field>
<field valueprop="weight" ...></field>
<field valueprop="birthday" ...></field>
```

The above field definitions are bound to the following set/get methods:

```
public void setName(String value) { ... }
public String getName() { ... }

public void setAge(int value) { ... }
public String getAge() { ... }

public void setWeight(float value) { ... }
public float getWeight() { ... }

public void setBirthday(Cdate value) { ... }
public Cdate getBirthday() { ... }
```

The correct property name starts with a lowercase letter, because the first letter is always converted to lowercase. Example:

```
<field valueprop="cAPITAL" ...></field>
```

The above field definition is bound to the following set/get method:

```
public void setCAPITAL(String value) { ... }
public String getCAPITAL() { ... }
```

Simple Properties which are Provided by Embedded Objects of the Adapter

Properties can also be provided by an embedded object of the adapter. The embedded object itself must be accessible by a corresponding get method.

Example:

```
<field valueprop="address.street"></field>
```

The above field definition points to a value which is provided in the following way:

```
public class XYZAdapter
    extends com.softwareag.cis.server.Adapter
{
    // access in the adapter to the address object
    public Address getAddress() { ... }
}

public class Address
{
    public String getStreet() { ... }
    public void setStreet(String value) { ... }
}
```

You can build any chaining of properties you desire.

As shown in the example, embedded objects need not be adapter objects. Only the root object is required to be an adapter.

Array Properties which are Provided Directly by the Adapter

You can use array properties and can access them directly within your binding definitions. An array property always returns an array of objects, each object providing either simple properties or array properties. The type of the object array is not relevant for the Application Designer runtime. If you just return "Object[]" as a result of the method, this is sufficient.

Example:

```
<field valueprop="addresses[0].street" ...></field>
```

The above field definition points to a property which is implemented in the following way:

```
public class XYZAdapter
    extends com.softwareag.cis.server.Adapter
{
    // access in the adapter to the address object
    public Address[] getAddresses() { ... }
}

public class Address
{
    public String getStreet() { ... }
    public void setStreet(String value) { ... }
}
```

Note that the name used inside the control definition for binding (`addresses[0].street` in the our example) can either be entered manually or is implicitly created by some controls. Example: in a TEXTGRID control, specify an array property for the entire control and a simple property inside the COLUMN definition:

```
<textgrid arrayprop="addresses" ...>
    <column property="street" ...></column>
    <column property="city" ...></column>
</textgrid>
```

The TEXTGRID control itself uses these definitions to ask for the properties `addresses[0].street`, `addresses[0].city`, `addresses[1].street`, `addresses[1].city` etc. at runtime.

Note that it is not possible to access an array of simple objects directly. It is not possible to define a field in the following way

```
<field valueprop="streets[0]"></field>
```

having a method:

```
public String[] getStreets() { ... }
```

You always have to go through an array of objects where each element itself provides access to simple properties.

Array Properties which are Provided by Embedded Objects of the Adapter

You can use any combination of *Simple Properties which are Provided by Embedded Objects of the Adapter* and *Array Properties which are Provided Directly by the Adapter*.

Example: define access to array properties in the following way:

```
<field valueprop="person.addresses[0].street" ...></field>

<textgrid arrayprop="person.addresses" ...>
  <column property="street" ...></column>
  <column property="city" ...></column>
</textgrid>
```

Access Path Restrictions

At runtime, Application Designer transfers the data from the adapter to the client. For accessing the data, it uses the following strategy:

- It asks the adapter object for all properties. This means, it calls all get methods that are defined as public methods.
- If the get method returns a simple value, is marked to be transferred. (Whether it is really transferred, depends also on the delta management between the client and the server.)
- If the get method returns an object (e.g. an address object as used in the previous sections) or an array of objects, these objects are used for further drill down.

This mechanism is flexible on the one side, but dangerous on the other side: the Application Designer runtime will load *all objects* by following up the get methods.

Consequently, there is a certain access path restriction inside the Application Designer environment: if you generate a page (either by the Layout Painter or by the logical interfaces to the HTML generator) an access path restriction file is generated in addition. The HTML generator parses all tags of a page; the controls themselves are bound to properties. This information is collected and written to a file.

This file is stored in the directory */accesspath* below the project directory. Please have a look at the files generated implicitly with your pages: the file contains a list of all access paths that are valid to be followed by runtime.

The name of the access path file is the same as the name of the page, but has the extension *.access*. Be aware of the fact that this access path file is inevitably important to avoid “mass loading” of data. Therefore, it must be a part of your software deployment.

27

Dynamic Access Property Binding

■ Interface IDynamicAccess	138
■ Example	138

Dynamic access binding is an additional binding technique that can be used together with **Java Bean binding** as described in the previous section. Dynamic access binding does not require an explicit definition of a set/get method for each property but is able to access the properties by generic data access functions.

Adapter objects, as well as embedded objects that are accessed inside the access path, may optionally support the interface `IDynamicAccess`. In this interface, you declare that there are additional properties to be accessed by generic access methods.

Interface `IDynamicAccess`

The interface definition looks as follows:

```
public interface IDynamicAccess
{
    public String[] findDynamicAccessProperties();
    public Class getClassForProperty(String property);
    public void setPropertyValue(String propertyName, Object value);
    public Object getPropertyValue(String propertyName);
    public void invokeMethod(String methodName);
}
```

It informs which dynamic properties are supported. In addition, you have to specify which class of a property it is. You can use any of the classes that are listed in *Appendix C - Data Types to be Used by Adapter Properties* for simple-value properties - and any class you desire for embedded object properties that you follow inside your access path. If you return a null value as a result of the `getClassForProperty()` method, the runtime returns the value as a String object.

Besides, you have to implement the generic set and get functions for property access.

There is a generic method invoked, e.g. when the user chooses a button in the page bound to a dynamically called method.

Example

The following example shows an adapter object that has two Bean properties (`firstName`, `lastName`) and three dynamic properties (`street`, `city`, `birthday`):

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.util.CDate;

public class DynamicAccess_Adapter
    extends Adapter
    implements IDynamicAccess
{
    String m_firstName;
    String m_lastName;
    String m_street;
    String m_city;
    CDate m_birthday;

    public String[] findDynamicAccessProperties()
    {
        return new String[] {"street","city","birthday"};
    }

    public void setPropertyValue(String propertyName, Object value)
    {
        if (propertyName.equals("street")) m_street = (String)value;
        else if (propertyName.equals("city")) m_city = (String)value;
        else if (propertyName.equals("birthday")) m_birthday = (CDate)value;
        else throw new Error("No property " + propertyName + " available");
    }

    public Object getPropertyValue(String propertyName)
    {
        if (propertyName.equals("street")) return m_street;
        if (propertyName.equals("city")) return m_city;
        if (propertyName.equals("birthday")) return m_birthday;
        throw new Error("No property " + propertyName + " available");
    }

    public Class getClassForProperty(String propertyName)
    {
        if (propertyName.equals("birthday")) return CDate.class;
        //      default: null ==> String is assumed by runtime
        return null;
    }

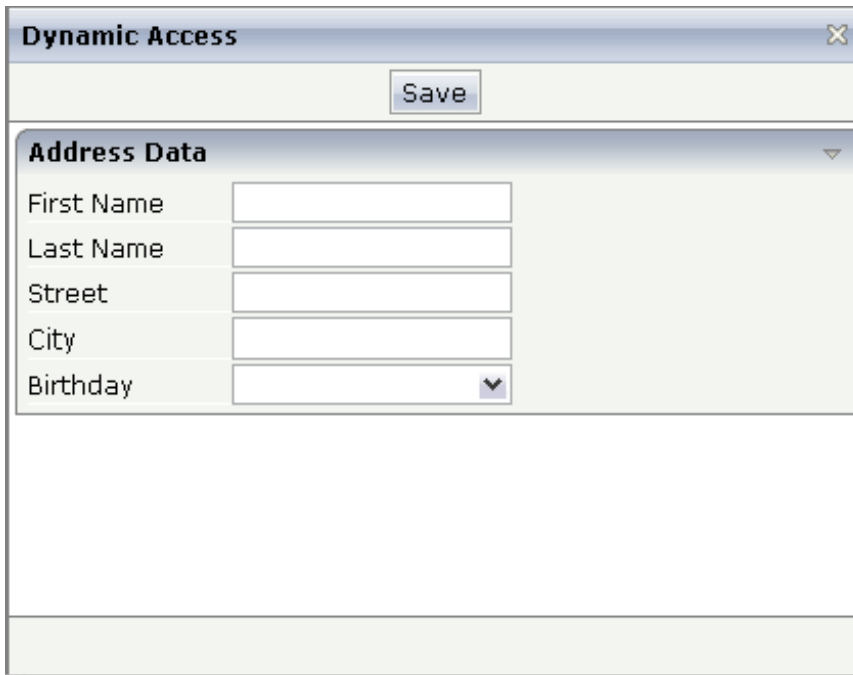
    public void invokeMethod(String methodName) {}

    // -----
    // bean properties
    // -----
    public void setFirstName(String value) { m_firstName = value; }
    public String getFirstName() { return m_firstName; }
    public void setLastName(String value) { m_lastName = value; }

```

```
public String getLastName() { return m_lastName; }  
}
```

Specifying a layout definition, there is no difference between the dynamic access properties and the bean properties.



The layout definition for the above page looks as follows:

```
<page model="DynamicAccess_Adapter">  
  <titlebar name="Dynamic Access">  
  </titlebar>  
  <header withdistance="false">  
    <button name="Save">  
    </button>  
  </header>  
  <pagebody>  
    <rowarea name="Address Data">  
      <itr>  
        <label name="First Name" width="100">  
        </label>  
        <field valueprop="firstName" length="20">  
        </field>  
      </itr>  
      <itr>  
        <label name="Last Name" width="100">  
        </label>  
        <field valueprop="lastName" length="20">  
        </field>  
      </itr>  
    </rowarea>  
  </pagebody>  
</page>
```

```
<itr>
  <label name="Street" width="100">
  </label>
  <field valueprop="street" length="20">
  </field>
</itr>
<itr>
  <label name="City" width="100">
  </label>
  <field valueprop="city" length="20">
  </field>
</itr>
<itr>
  <label name="Birthday" width="100">
  </label>
  <field valueprop="birthday" length="20" datatype="date">
  </field>
</itr>
</rowarea>
</pagebody>
<statusbar withdistance="false">
</statusbar>
</page>
```


28 XML Property Binding

Use XML property binding with the following:

- ICONLIST control,
- MENU control,
- ROWTABSUBPAGES control, or
- for any simple property.

XML property binding uses XML files to access property values. Use the prefix "XML:" to indicate XML property binding.

```
<itr visibleprop="XML:isHomeAddressVisible">
    ...
</itr>
```

You see that the visibility of the row container is controlled by the XML property `isHomeAddressVisible`. An XML property is bound to a property tag (name-value pair).

```
<property name="isHomeAddressVisible" value="true">
</property>
```

The overall page layout is bound to an XML data file that contains all the property tags.

```
<xmlproperties>
  <property name="isHomeAddressVisible" value="true">
  </property>
  <property name="isBusinessAddressVisible" value="false">
  </property>
</xmlproperties>
```

The XML data file contains two property tags. With the first property, `isHomeAddressVisible` is set to "true"; with the second property, `isBusinessAddressVisible` is set to "false". At runtime,

you can switch between XML data files by changing the “XML data mode”. Just use the following method in order to use the correct XML data file method:

```
Adapter.setXMLDataMode
```

The files are kept within directory `<webapp>/<project>/xmldata`. Each XML data mode is represented by a subdirectory. By default, the Application Designer server accesses the XML files within the directory *default*.

```
cis
  project
    xmldata
      default
        PersonInfoAdapter.xml
      fullinfo
        PersonInfoAdapter.xml
```

29

Getting Information about Access Paths

Sometimes you need to get detailed information about a page accessing its adapter. Or, in other words: you want to get a detailed list of all the properties and objects which are referenced by your page definition to the corresponding adapter object.

For this reason, there is the class `CheckAccessPath` in the `com.softwareag.cis.server` package providing this information. The class has a `getInstance()` method to obtain an instance; the class has two additional methods:

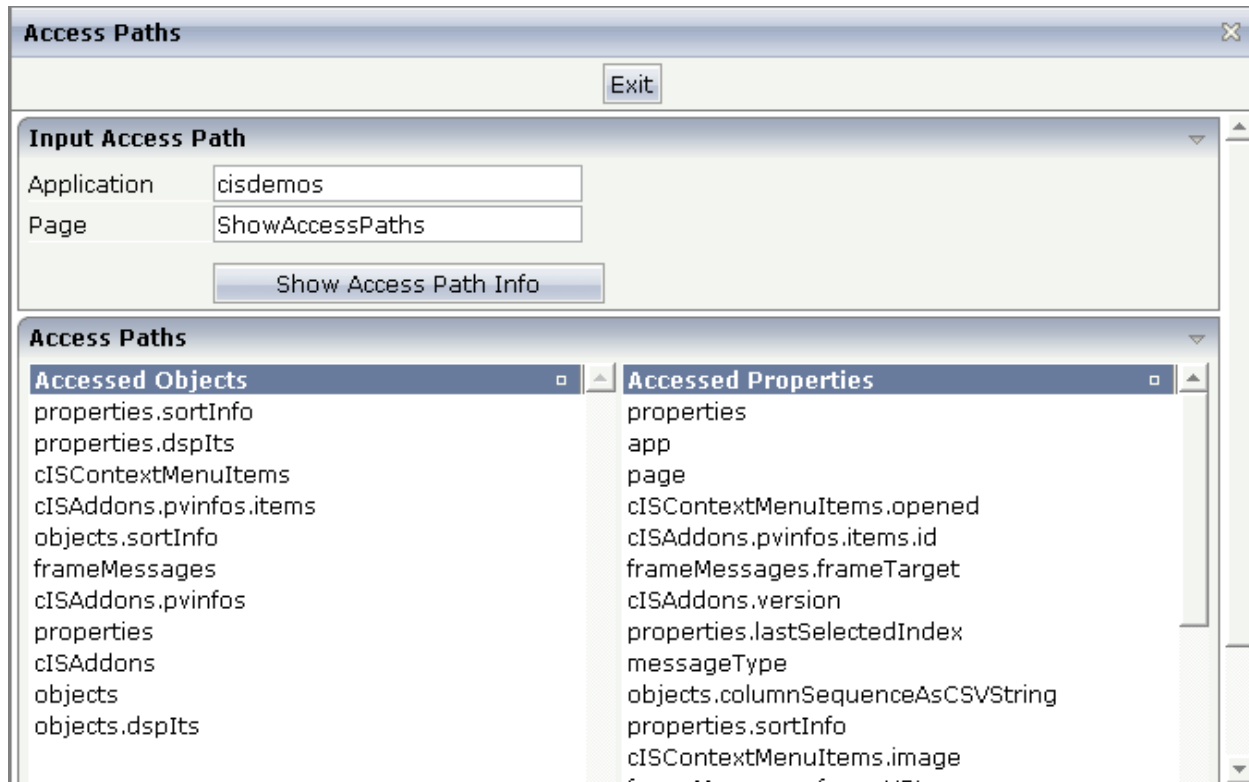
```
public String[] findAccessedObjects(String application,
                                   String reference);
public String[] findAccessedProperties(String application,
                                      String reference);
```

In both methods, the parameters are "application" and "reference". "application" is the application project in which a page is defined. "reference" is the name of the page itself, without ".xml".

The `findAccessedObjects` method returns a `String` array of all objects referenced in this page. An object is referenced if the properties are not directly plugged to the adapter itself but to subobjects. Example: if you bind a `FIELD` to the `VALUEPROP` "address.street" then "address" is the returned name.

The `findAccessedProperties` method returns a `String` array of all properties referenced in the page which are not complex properties, but simple value properties.

In the `cisdemos` project (which is part of the installation), there is a page "ShowAccessPaths" and the corresponding class `ShowAccessPathsAdapter` that shows an example of how to use the `CheckAccessPath` methods. The page allows you to enter the application name and the page name, and returns a list of referenced objects and properties:



In the class definition of the corresponding adapter, the code (finding the information about access paths) looks as follows:

```
// -----
// inner classes
// -----
public class Info
{
    // property >objects[*].name<
    // property >properties[*].name<
    String m_name;
    public String getName() { return m_name; }
    public void setName(String value) { m_name = value; }
}

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

// property >app<
String m_app = "";
public String getApp() { return m_app; }
public void setApp(String value) { m_app = value; }

// property >page<
String m_page = "";
```

```

public String getPage() { return m_page; }
public void setPage(String value) { m_page = value; }

// array property >objects[*]<
TEXTGRIDCollection m_objects = new TEXTGRIDCollection();
public TEXTGRIDCollection getObjects() { return m_objects; }

// array property >properties[*]<
TEXTGRIDCollection m_properties = new TEXTGRIDCollection();
public TEXTGRIDCollection getProperties() { return m_properties; }

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

/** */
public void onShowAccessPath()
{
    // check
    if (m_app.trim().length() == 0)
    {
        outputMessage(MT_ERROR, "Please specify application project");
        return;
    }
    if (m_page.trim().length() == 0)
    {
        outputMessage(MT_ERROR, "Please specify page");
        return;
    }
    // fill data
    m_properties.clear();
    m_objects.clear();
    String[] objects = ↵
CheckAccessPath.getInstance().findAccessedObjects(m_app, m_page);
    String[] properties = ↵
CheckAccessPath.getInstance().findAccessedProperties(m_app, m_page);
    for (int i=0; i<objects.length; i++)
    {
        Info info = new Info();
        info.m_name = objects[i];
        m_objects.add(info);
    }
    for (int i=0; i<properties.length; i++)
    {
        Info info = new Info();
        info.m_name = properties[i];
        m_properties.add(info);
    }
}
}

```


30

Exception Management Inside an Adapter Object

■ Normal Exceptions are to be Handled by the Application	150
■ Errors and Runtime Exceptions - The Default Behavior	151
■ Interrupting the Application Designer Request Processing - AdapterNotAvailableError	152
■ Errors and Runtime Exceptions - The Special Behavior	317

Application Designer binds its page processing to adapter objects providing properties and methods - as explained in the previous sections. What happens if an error happens at runtime, e.g. an error occurs in the method of an adapter object that is called after the user pressed a button?

Normal Exceptions are to be Handled by the Application

The first rule is: normal exceptions (i.e. no “Errors”, no “Runtime Exceptions”) are to be handled by the application itself.

This means: a property is provided by the corresponding set/get methods (or by an equivalent method when using dynamic binding). The methods must not throw any exception, i.e. in their declarations there is no "throws" element.

Example for a correct implementation:

```
public void setFirstName(String value)
{
    ...
    ...
}
public String getFirstName()
{
    ...
    ...
}
```

The following example is an *incorrect* implementation because application exceptions are thrown:

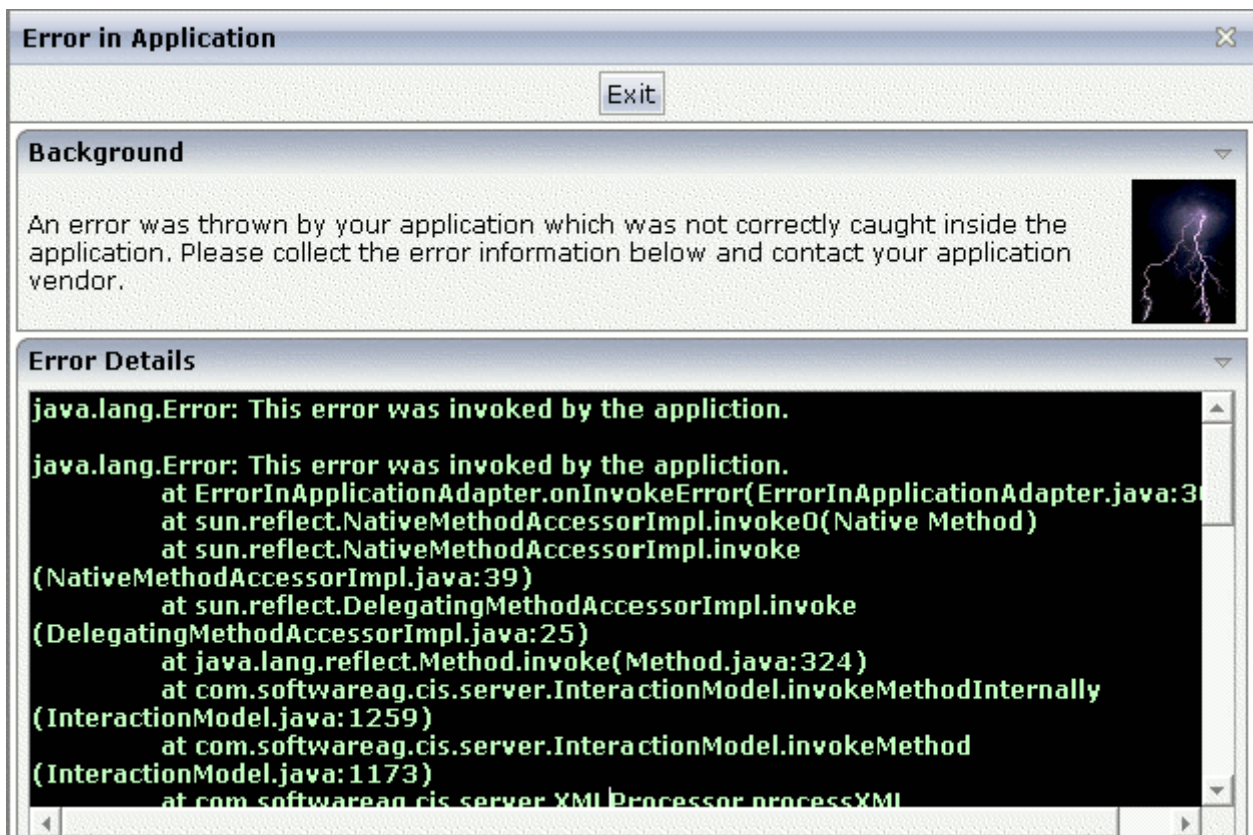
```
public void setFirstName(String value)
    throws ApplicationException
{
    ...
    ...
}
public String getFirstName()
    throws ApplicationException
{
    ...
    ...
}
```

Consequence: Application Designer passes values from the browser front end into the adapter object, invokes certain activities inside this object and collects data from the object to pass data changes back to the browser. Application exceptions are not relevant from Application Designer's point of view - they only affect the application internally.

Errors and Runtime Exceptions - The Default Behavior

Of course your application still can throw “Error” exceptions or “Runtime” exceptions. These are the exceptions that need not be declared inside a method’s code - but that can be thrown any time at any place.

If Application Designer receives an error or runtime exception, Application Designer displays a page by default in which the error information is shown.



In addition, Application Designer writes a full stack dump into its runtime log.

Interrupting the Application Designer Request Processing - AdapterNotAvailableError

There may be some situations - within a special environment context - that do not allow to process the page at all. Maybe you have a page that requires a user to be logged on; if the Application Designer request processing starts now, you may decide with the method `reactOnDataCollectionStart` that you do not want to start the request since it does not make sense at all and just causes exceptions.

The only thing you want to do in such a scenario is to “escape” to a page which helps out of the situation. For example, if you miss logon information, you want to “escape” to the logon page and return to your original page afterwards.

The Java error class `AdapterNotAvailableError` is provided for this reason. The following adapter code shows an example on how to handle this error:

```
package com.softwareag.cis.demoapps;

...
...

public class Rescue1Adapter
    extends Adapter
{
    ...
    ...
    /** start of data transfer */
    public void reactOnDataTransferStart()
    {
        super.reactOnDataTransferStart();
        // fetch user and sytem info from session context
        m_user = (String)findSessionContext().lookup("rescueexample/user",false);
        m_system = (String)findSessionContext().lookup("rescueexample/system",false);
        // if not logged on ==> switch to rescue2 page
        if (m_user == null ||
            m_system == null)
        {
            // prepare Rescue2 page
            Rescue2Adapter r2a = (Rescue2Adapter)findAdapter(Rescue2Adapter.class);
            r2a.init("Rescue1.html");
            // throw error in order to interrupt normal processing and switch
            // to Rescue2 page
            throw new AdapterNotAvailableError("Rescue2.html");
        }
    }
    ...
}
```

```
...
}
```

Inside the `reactOnDataTransferStart` method, a user and a system variable are read from the session context. If one of them is null, the adapter decides to switch to page *Rescue2.html* and throws an `AdapterNotAvailableError` error. Before, it pre-fetches the page adapter of the page to escape to and initializes the page with certain information (in this example, it passes its own page name).

The error page is opened inside the same subsession as the one throwing the error.

Errors and Runtime Exceptions - The Special Behavior

There is a set of methods available in the adapter with which you can influence the standard error behavior:

- `handleErrorDuringInitPhase()`
- `handleErrorDuringSetPhase()`
- `handleErrorDuringInvokePhase()`
- `handleErrorDuringGetPhase()`

Depending on the method you have the following possibilities:

- `handleErrorDuringInitPhase()`
This method is called when an error occurs in the `init` method of the adapter.
- `handleErrorDuringSetPhase()` **and** `handleErrorDuringGetPhase()`
These methods are called when an error occurs during the set and the get phase of the adapter request processing.

You may throw an `AdapterNotAvailableError` to navigate to a page of your own in order to present to the user detailed error information - and maybe also some way to solve the error.

- `handleErrorDuringInvokePhase()`
This method is called when an error occurs during the invoke phase of the adapter request processing.

You can decide whether normal Application Designer runtime processing continues, whether you want to navigate to an error handling page (via `PageNotAvailableError`), or whether the standard error processing of Application Designer is done.

See the API documentation (Java Doc) for further details.

31

Additional Interfaces

■ Extending the Set of Simple Data Types	156
■ Avoid the Getting of Certain Simple Data Type Properties	157
■ Exchanging Objects by Converter Objects	158

Some additional interfaces are available which allow you to modify the binding behavior between a page and its adapter object. The interfaces are available via `com.softwareag.cis.server.IInteractionSessionMgr` which represents a general interface to the Application Designer runtime.

You receive an instance of `IInteractionSessionMgr` in the following way:

```
...  
...  
IInteractionSessionMgr iism = InteractionSessionMgrFactory.getInteractionSessionMgr();  
...  
... ↵
```

Extending the Set of Simple Data Types

As described previously in this part, Application Designer collects all the properties of an adapter object (and its contained objects) when collecting the data in order to respond to the browser client.

The way Application Designer collects the data for a certain object is:

- Application Designer collects all the properties that represent simple data types (int, float, String, BigDecimal, CDate, etc.; see *Appendix C - Data Types to be Used by Adapter Properties*).
- Application Designer investigates all properties that are non-simple datatypes and that are part of the access path of a certain page.

Sometimes you want to add a certain class to be managed as “Simple Datatype Class”, i.e. Application Designer will not treat objects of this class as non-simple objects but will treat them as simple objects.

Simple objects have to provide a class implementation that

- provides a constructor in which the value is passed as a string object, and
- provides a `toString()` method to get the String representation of the contained value.

Example of a valid class:

```
public class ExtendedString  
{  
    String m_value;  
    public ExtendedString(String value)  
    {  
        m_value = value;  
    }  
    public String toString()  
    {
```

```

        return m_value;
    }
    ...
    ...
}

```

The class is registered by using the method `IInteractionSessionMgr.registerPropertyAccessSimpleDatatypeExtension()`:

```

IInteractionSessionMgr iism;
iism = InteractionSessionMgrFactory.getInteractionSessionMgr();
iism.registerPropertyAccessSimpleDatatypeExtension(ExtendedString.class);

```

Now having an adapter object (or follow-on object such as grid item) providing a property of type `ExtendedString`, Application Designer will not drill down the object but will use the object's `toString()` method to get its value and will use the object's constructor to pass new values to the application.

Avoid the Getting of Certain Simple Data Type Properties

In the previous sections, the general rule was explained: if Application Designer investigates an object during the get-data phase, then

- it takes all simple data type properties, and
- it takes those complex data type properties that are required by the corresponding page.

There is one possibility to fine-control the getting of simple data type properties: Every object that is investigated by Application Designer during the get phase (e.g. the adapter object) can implement the interface `com.softwareag.cis.server.IControlPropertyAccess`. The interface is defined as follows:

```

public interface IControlPropertyAccess
{
    public String[] findPropertiesNotToBeCollected();
}

```

When the interface is implemented, the get methods that are passed back by the `findPropertiesNotToBeCollected()` method are not processed.

Note that the method is called once per class - the first time Application Designer interacts with an object. You cannot tell Application Designer by this interface to sometimes use the property and sometimes not.

Exchanging Objects by Converter Objects

When Application Designer is accessing properties that are non-simple data type objects, there is the possibility to exchange the object and tell Application Designer to use a converter object instead.

The interfaces are:

- With `IInteractionSessionMgr.registerPropertyAccessConverter(Class forClass, IPropertyAccess Converter converter)` you can register a class (parameter `converter`) that is used as converter for another class (parameter `forClass`).
- The converter class itself must support the interface `IPropertyAccessConverter` that looks as follows:

```
public interface IPropertyAccessConverter
{
    public Object getConvertedObject(Object propertyValue);
}
```

For more details, see the JavaDoc API documentation.

V

Details on Session Management

In *Working with Page Navigation*, there is a brief description on how Application Designer manages sessions. This part provides more details about session management.

In principle, the session management is hidden inside Application Designer. If you write normal applications running in the Application Designer workplace environment, you do not have to care about session management at all: you do not have to somehow collect data from a session object in order to work with it or do something similar.

However, reading this part is interesting for you if you want to know the following:

- What is the life cycle of an adapter?
- What amount of data is kept in an adapter?
- How does Application Designer internally arrange adapters?

This part is especially important for you if you:

- write a workplace-like application which serves as a frame for content applications;
- not only have Application Designer pages in your web application but also other servlets or JSP pages.

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

[HTTP Sessions - Application Designer Sessions](#)

[Application Designer Session - Application Designer Subsessions](#)

[Application Designer Subsession - Application Designer Adapter Objects](#)

[How Things Start](#)

[How Things End](#)

[Workplace Management](#)

[Saving Context Data](#)

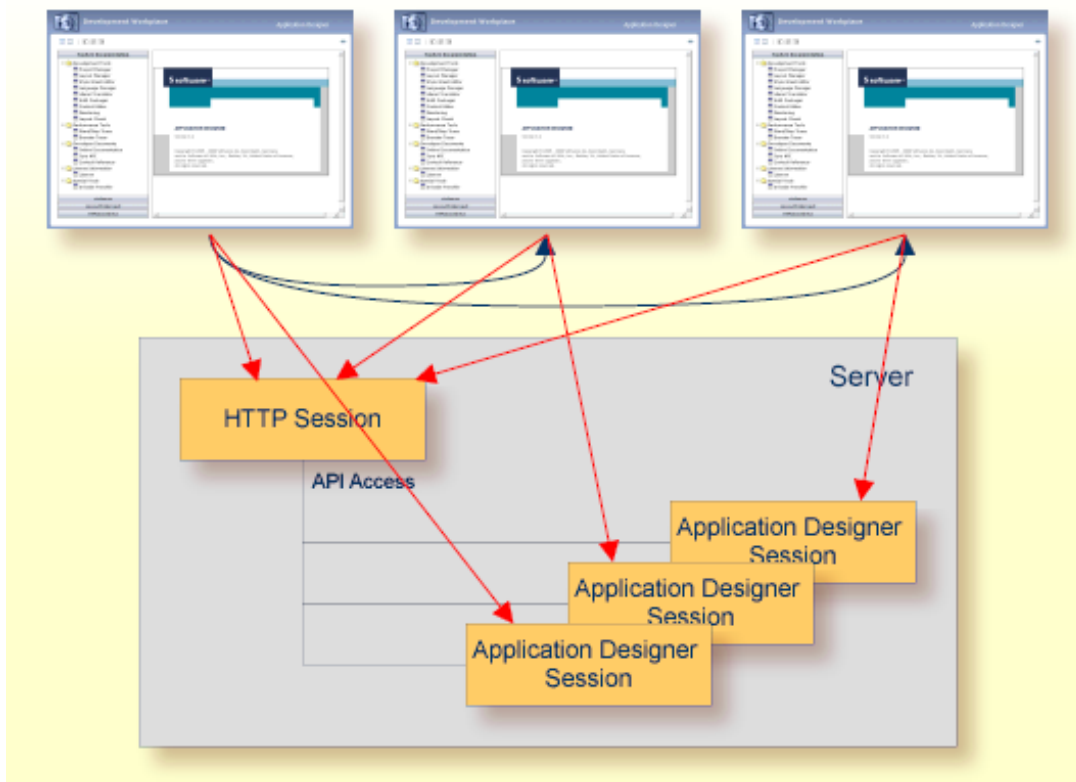
[Session IDs](#)

32 HTTP Sessions - Application Designer Sessions

If you have already developed servlets/JSPs, your first question will be: how do Application Designer sessions relate to HTTP sessions?

Application Designer adapters are living in sessions which are administered inside the Application Designer runtime environment. The sessions are kept in parallel to HTTP sessions, i.e. HTTP sessions may be used by other servlets/JSPs that may be part of your web application - but Application Designer itself does not require them. It is no problem to reach HTTP sessions from an adapter object via an API.

Why is Application Designer not using straight HTTP sessions? The problem is that HTTP sessions are sometimes the same for multiple browser instances. If you open a new browser instance from an existing browser instance (for example, with the Internet Explorer), then the corresponding session object on the server is shared between the browser instances. In the Application Designer session management, each instance of a browser (and if you want: each frame inside one browser) has its own clearly assigned session.



The above diagram shows the following:

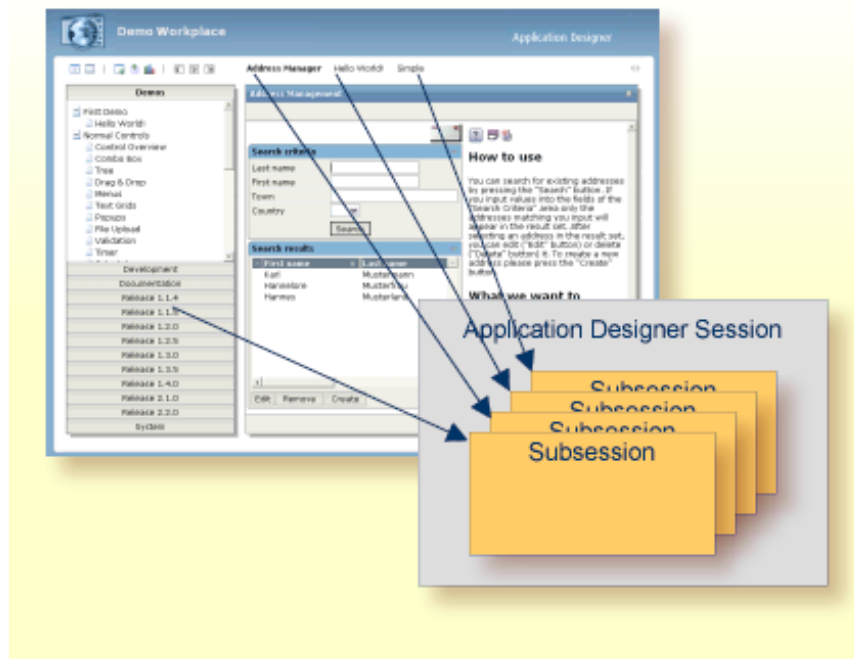
- There are three browser instances sharing one HTTP session.
- Each browser instance has one related Application Designer session.
- There is an API from the Application Designer runtime to access the HTTP session.

33 Application Designer Session - Application Designer

Subsessions

The Application Designer session concept knows one level below the Application Designer session: the Application Designer subsession. Adapter objects are living inside one subsession - and there can be multiple subsessions within one session.

Let us approach the subsessions by a practical example:

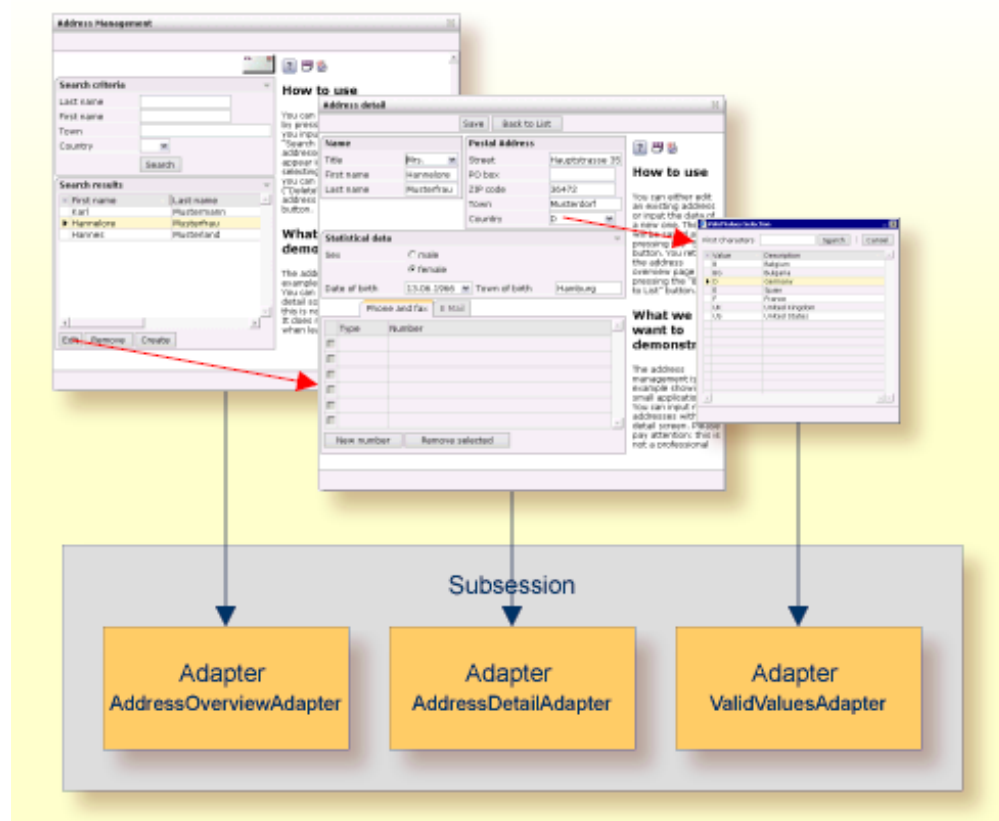


In the diagram, the Application Designer demo workplace is shown. Inside the workplace, three activities have been opened. The "Address Management" application is currently active. In addition, the workplace is also running.

See the next section [*Application Designer Subsession - Application Designer Adapter Objects*](#) for further information on this example.

34 Application Designer Subsession - Application Designer Adapter Objects

Each of the activities listed in the previous section *Application Designer Session - Application Designer Subsessions* is represented by a subsession on the server side. Each subsession itself is holding the adapters for the activity.



In the diagram above, the activity "Address Management" is shown in detail. It consists of several pages between which the user navigates. Each page belongs to one adapter of a certain class. The adapter instances are managed inside the subsession.

The general rules for administering adapter instances are:

- For each adapter class, there is one instance inside one subsession. This means: if you have several different pages between which you navigate inside one activity, and all pages are bound to the same Adapter class, then all pages are working with the same server side adapter instance.
- Adapter instances start to live when they are first accessed (e.g. by a page requesting them). They are kept as instances for the whole life cycle of a subsession - if not explicitly destroyed by the application via an API call.

Basically, there are two types of sessions:

- Each browser connected to Application Designer opens a new session inside the server. When closing the browser or navigating to another web page, this session is automatically destroyed at the server side.
- Within a session there are subsessions. Each subsession represents the state of one interaction process inside the browser. In the Application Designer workplace environment, you can open multiple parallel interaction processes, and you can switch from one to another. You may have other environments in which you do not want to offer the multi-interaction process management - and only have one subsession for the whole life cycle of a session.

Inside a subsession, the adapter instances are created. All navigation is done between pages that belong to the same subsession. See also *Working with Page Navigation*.

35

How Things Start

- Starting an Application Designer Session 168
- Starting Additional Application Designer Subsessions 168

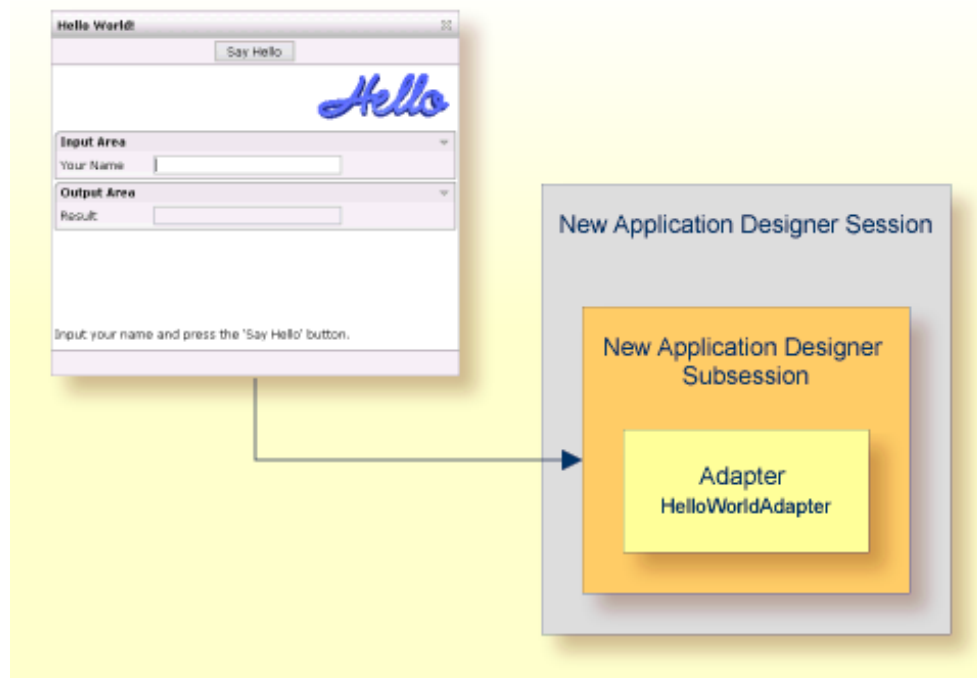
Starting an Application Designer Session

The proper start of a session is to open an Application Designer page via the StartCISPage servlet.

Example: If you start the "Hello World!" page with the following URL, a new Application Designer session object with a new session ID is automatically created on the server side:

```
http://localhost:51000/cis/servlet/StartCISPage?PAGEURL=/cisyourfirstproject/helloworld.html
```

The logical counter part of the page - the HelloWorldAdapter object - is opened inside a subsession that is automatically created inside the Application Designer session.



You see that inside one Application Designer session, there is always at least one subsession.

Starting Additional Application Designer Subsessions

You may use your pages in a mode in which you always work inside one Application Designer subsession - the one which was created during the StartCISPage procedure. But maybe you want to start additional subsessions.

There are two good reasons for starting additional subsessions:

■ **Separate life cycles of activities**

A subsession is keeping all adapter instances which play a role inside the processing of a certain activity. By closing the subsession, all adapter objects belonging to the subsession are released and can be caught by the garbage collector. In other words: a subsession is something like the life cycle manager for its contained adapters.

Consequence: if you have multiple activities running in parallel, then each activity has its own life cycle, e.g. it can be closed individually without any consequence for the life cycle of the other activities.

■ **Isolated activities**

The adapter objects are created per subsession. This means: you can run one and the same activity in parallel - represented by two subsessions. In both subsessions, adapters are built up in parallel - completely isolated from one another.

Programming applications inside “multi document interface”-like programs, (e.g. applications inside the Application Designer workplace) is therefore simple: each document (activity) is associated with its own subsession. The workplace just coordinates that the correct page is linked to the correct subsession at the appropriate point of time.

The starting of a new Application Designer subsession is done by opening a page inside a frame or inside an Application Designer subpage via Application Designer APIs.

Application Designer offers APIs (in class `com.softwareag.cis.server.Adapter`) to open Application Designer pages in a certain frame. These APIs always have one “simple” variant and one “complex” variant:

■ **Simple Variant**

```
protected void openCISPageInTarget(String pageURL,
                                   String target)
```

By calling this method, you open a certain page in a certain frame. The page is automatically linked to the subsession of the adapter calling this method.

■ **Complex Variant:**

```
protected void openCISPageInTarget(String pageURL,
                                   String subsessionId,
                                   String target)
```

By calling this method you open a certain page in a certain frame. But now you can explicitly pass a new `subsessionId` to be used for the page's adapter.

The proper call for a page which should belong to a new subsession is:

```
...  
...  
public void onOpenNewPage()  
{  
    // create new subsession id  
    String newSSID = UniqueIdMgmt.createPseudoGUID();  
    openCISPageInTarget("...URL...",newSSID,"...TARGET...");  
}  
...  
...
```

36

How Things End

■ End of an Application Designer Session	172
■ End of an Application Designer Subsession	172
■ End of an Application Designer Adapter	172

End of an Application Designer Session

A session normally ends if the page which was opened with the `StartCISPage` servlet is closed. This happens for example:

- if the user shuts down the browser,
- if the user loads a new page into the frame in which the `StartCISPage` servlet was called previously.

In other words: the session is normally kept alive as long as the user stays in the Application Designer environment.

Why “normally”? If a session is without user interaction for a long time, the session is timed out on the server side. When the user comes back to continue interaction, a corresponding message appears. The duration until a session is timed out is configurable; see the description of the *cisconfig.xml* file in the *Configuration* documentation for details.

If a session ends, all its subsessions and all adapters in the subsessions are automatically ended.

End of an Application Designer Subsession

A subsession is ended via an API. There are two APIs available:

- Via the interface `com.softwareag.cis.server.IInteractionManager`.
- Via the method `endProcess()` which your adapters inherit from the `Adapter` class.

For further information, see the JavaDoc documentation.

End of an Application Designer Adapter

Adapters typically stay alive until the subsession ends in which they are living. There is also an API available to directly end adapters:

- Method `Adapter.markThisAdapterForDestroy()`.
- Via the interface `IInteractionProcess` which you receive inside an adapter via `this.m_interactionProcess`.

37 Workplace Management

After reading the previous sections, you may now see in a better way what the task of a workplace management inside Application Designer is: a workplace is an application on its own having the task to administer content applications both from the graphical and the session management point of view.

The workplace management is responsible for the proper assignment of subsessions to activities. The life cycle of subsessions is typically controlled by the workplace.

38

Saving Context Data

■ Different Levels of Context	176
■ Accessing the Context	176
■ Typical Usage Scenarios	367

Sometimes it is useful to save context data centrally inside a session context and to use these data like a session-global variable. You should be very restrictive with this option - otherwise you may end up in a scenario in which any kind of data exchange is done by the context.

Different Levels of Context

The session management allows you to hold context information at two levels:

- **Session Context**

Within the session context, save data that you want to access from everywhere inside your adapters.

- **Subsession Context**

Within the subsession context, save data which you want to access from everywhere inside a subsession.

Two different subsessions have also two different subsession contexts, i.e. the saved data are kept independent per subsession.

Accessing the Context

You obtain the context(s) by calling methods which are inherited from the `Adapter` class:

- `findSessionContext()`

Returns a context which is held for each session.

- `findSubSessionContext()`

Returns a context which is held for each subsession.

Both methods return a `com.softwareag.cis.context.ILookupContext` interface. This interface offers the possibility to bind and look up any objects.

```
public interface ILookupContext
{
    public Object lookup(String s, boolean reactWithErrorIfNotExist);
    public void bind(String s, Object o);
    public void releaseAllReferences();
}
```

When binding objects to a context, use a naming convention that is similar to the naming of your Java classes to avoid naming conflicts. Example:

```
...  
findSessionContext.bind("com/yourcompany/application/parameter");  
...
```

The context can be cleaned up by the `releaseAllReferences()` method. It is integrated into Application Designer's session management.

Typical Usage Scenarios

Examples of typical data that you save at the session context level:

- Name of the user who is currently logged on.
- Name of the system to which the user is currently logged on.
- Language in which the user is logged on.

Examples of typical data that you save at the subsession context level:

- ID of the object you are processing.
- Temporary data you want to pass from one page to another.

39

Session IDs

Each session - session or subsession - holds an ID.

■ Session

The ID of the session is unique inside one instance of Application Designer. If you have two Application Designer installations running, the same ID may be used inside both servers.

■ Subsession

The ID of a subsession is unique inside one session. If you have multiple sessions running inside one Application Designer instance, the same subsession ID may be used in two sessions.

You can access the IDs from your adapter in the following way:

```
public class TestAdapter
    extends com.softwareag.cis.server.Adapter
{
    ...
    ...
    public void xxx()
    {
        ...
        String sessionId = this.m_interactionProcess.getSessionId();
        String subsessionId = this.m_interactionProcess.getProcessId();
        ...
    }
    ...
    ...
}
```


VI

Application Project Management

In the "Hello World!" example of the *First Steps*, you used the application project "cisyourfirstproject" to develop your first Application Designer application. This part provides more details on project management.

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

What is an Application Project?

Class Loading Issues

Application Project Directory

Application Project Context Root

Creating an Application Project

Tools for Application Project Management

40 What is an Application Project?

According to the information in the *Introduction*, Application Designer runs as a web application or as part of a web application inside a servlet engine.

If you have larger projects and consequently have a high number of layouts and adapter classes, then you want to structure your development activities in a better way. For this reason, a project management exists that you can use to separate your project's resources across so-called application projects.

An application project is represented in the file system by a single directory. It includes:

- XML layout definitions.
- HTML files which are generated from the layout definitions.
- Required images or required “normal” HTML files.
- Adapter classes.
- Files holding the information on how to translate text IDs into readable words in a language-dependent way.
- Help files.

You can (and should) arrange your application projects to be self-containing units.

41

Class Loading Issues

If you use Application Designer in design time mode (i.e. using the Application Designer class loader), then place the classes for each project inside the project's */appclasses/classes* or */appclasses/lib* directory.

Make sure that the same class does not occur in different projects. The Application Designer class loader embeds all classes of all projects in one view and loads the first class it finds. Keep in mind: if turning to runtime mode (i.e. class loading of web application class loader), then all classes have to be transferred to */WEB-INF/appclasses/classes* and */WEB-INF/appclasses/lib* anyhow, and having two class implementations holding the same names also will cause conflicts at this point of time.

For detailed information, see *Appendix D - Class Loader Concepts*.

42 Application Project Directory

Each application project is represented by a subdirectory of the web application's directory inside the Application Designer installation. If you stick to the default installation and you create an application project "appxyz", the application project directory is:

```
<install_dir>/tomcat/webapps/cis/appxyz/
```

The application project directory itself is subdivided into several other directories:

Directory	Description
<i>accesspath/</i>	This is where access restriction files are stored.
<i>appclasses/classes</i>	This is where the class loader looks for *.class files.
<i>appclasses/lib</i>	This is where the class loader looks for *.jar files.
<i>/multilanguage</i>	This is where the multi language management looks for its files.
<i>/xml</i>	This is where the XML layout definitions are stored.

HTML files (e.g. generated intelligent HTML pages) are stored directly in the application project directory.

43 Application Project Context Root

Each application project can be reached from the web server/servlet engine in the following way:

```
http://<host>:<port>/<WebAppName>/<ApplicationProjectName>
```

Taking the project "appxyz" from above and still sticking to the standard installation, the URL looks as follows:

```
http://localhost:51000/cis/appxyz
```


44 Creating an Application Project

You create an application project using the Project Manager which is available in Application Designer's development workplace. For detailed information, see *Project Manager* in the *Development Workplace* documentation.

If you use the standard Tomcat servlet engine, you have to restart Tomcat after having created the new application project.

45

Tools for Application Project Management

After having created a project, certain tools are available in which the project becomes visible.

The navigation frame of the development workplace shows one button for each project. In each topic, you see the existing layouts for the project.

When you open the Layout Manager, the application project becomes selectable in the list of projects.

VII

Dynamic Page Layout

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

Introduction

Scenarios

Dynamic Pages - Normal Pages

Programming Dynamic Pages

Interface IDynamicPageMgmt

Background Information

Dynamic Pages - Dynamic Adapters

46

Introduction

There are three ways to provide for a dynamic page layout:

- Some controls support dynamic rendering that can be controlled by corresponding adapter properties.

A FIELD control, for example, can be influenced by adapter properties that control whether the field is editable or not. A BUTTON control can refer to an adapter property telling whether the button is visible or not. There is a ROWDYNAVIS control that represents a container area that can be switched on/off by your server side adapter. When switched on or off, the whole container is visible or invisible.

- If a page gets too large, you usually split up the page into logical areas, each having a certain level of independency.

With the SUBCISPAGE2 control, you can embed one screen flexibly into other screens. The same can be done by using the ROWTABSUBPAGES control. You can also distribute your layout across multiple frames of a frameset. This is, for example, done inside the workplace itself: the content frame of the workplace is dynamically started - it displays the activity which is currently active.

- Screens can be built up in a “100% dynamic way”. This means that the layout that you normally create when designing a page is defined at runtime by a program.

Information on the first two options is provided in *Working with Controls* (in the *Layout Elements* documentation), and in *Embedding Pages into Pages* and *Embedding Pages into a Workplace* (in the *Working with Pages* documentation).

This part tells you about the last option: building dynamic pages.

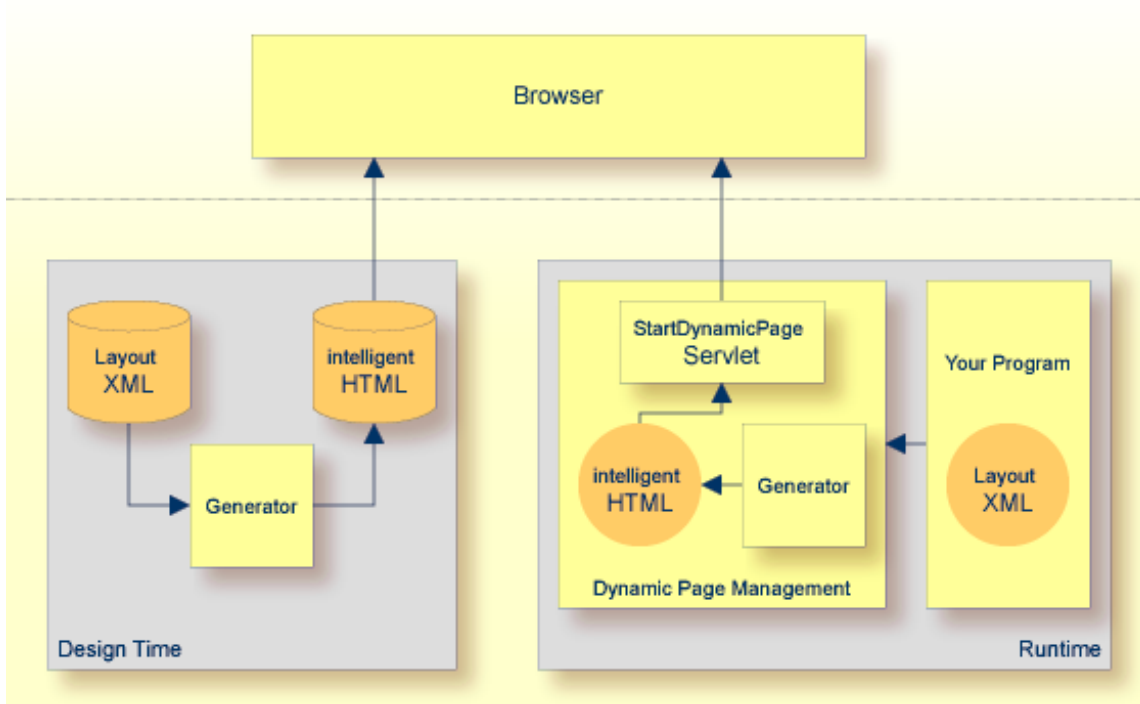
47

Scenarios

You use dynamic pages in scenarios in which it is not possible to define the layout of a page at design time. Typically you are in the process of developing a generic part of your application when desiring dynamic pages.

Example: you want to create a generic application for maintaining table records of a database. You may have a detail screen for one table record: the number of fields and their interrelation depends on the table. In order to provide a properly usable screen, you can generate the detail screen dynamically - based on meta data which comes from the database management system.

Dynamic pages follow the same principles as normal Application Designer pages. The following diagram explains the difference:



The normal way for creating intelligent HTML pages is to use the Application Designer design time environment: during the development process, XML layout definitions are translated into intelligent HTML pages. These pages are loaded and executed by the browser.

For the browser, it is not really relevant where the HTML behind a URL comes from - it does not need to know if it is coming from a static HTML file or if it is coming from a program. Consequently,

the dynamic page management just uses the components of the design time environment and puts them into the runtime context:

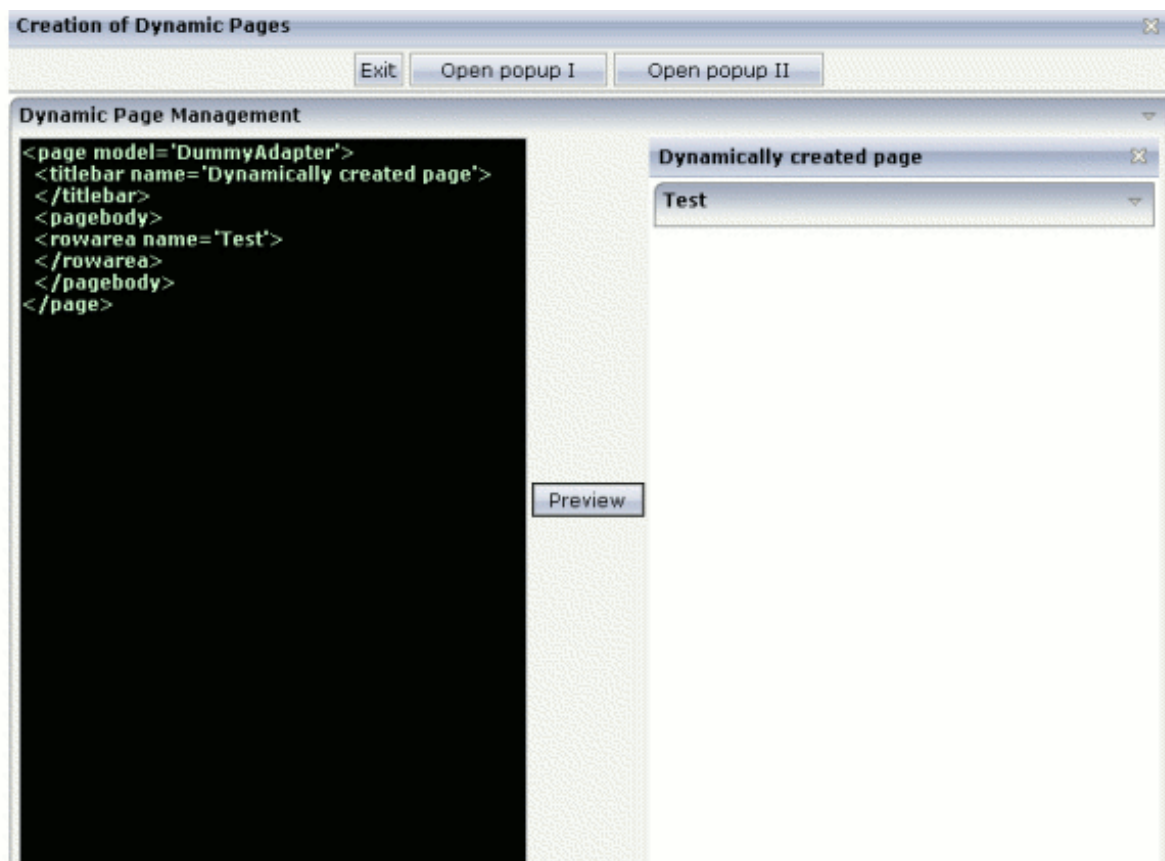
- Via the API `com.softwareag.cis.servcer.IDynamicPageMgmt`, you can pass XML layout that is translated into corresponding HTML code. When passing the XML, you define a logical name. The generated HTML is kept under this name.
- From now on, you can reference the generated HTML by using a servlet that receives as parameter the logical name of what you passed.

All steps are performed in memory, i.e. there is no file which is stored. The dynamically created page is kept in the context of one Application Designer session. After the session is destroyed, the dynamic page is removed from memory.

49

Programming Dynamic Pages

This section gives an example on how to program the dynamic page management.



On the left of the example page, you can enter any XML layout definition. When choosing the **Preview** button, the layout definition is translated into an HTML page. The page is shown in the right area.

The main controls that are used are:

- The TEXT control - for entering the text.
- The SUBCISPAGE control - for displaying the preview.

The XML layout definition looks as follows:

```
<rowarea name="Rowarea" height="100%">
  <itr takefullwidth="true" height="100%">
    <text valueprop="layoutXML" width="50%" height="100%"
      textareastyle="background-color:#000000;color:#D0FFD0;font-weight:bold">
    </text>
    <hdist>
    </hdist>
    <button name="Preview" method="onPreview">
    </button>
    <hdist>
    </hdist>
    <SUBCISPAGE valueprop="subpageURL" width="50%" height="100%" borderwidth="0">
    </SUBCISPAGE>
  </itr>
</rowarea>
```

The adapter code looks as follows:

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

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

public class dynamicPagesAdapter
  extends Adapter
{
  String m_layoutXML;

  public String getLayoutXML() { return m_layoutXML; }
  public void setLayoutXML(String value) { m_layoutXML = value; }

  String m_subpageURL = "/HTMLBasedGUI/empty.html";
  public String getSubpageURL() { return m_subpageURL; }
  int m_counter = 0;
  // -----
  // public adapter methods
  // -----

  public void onPreview()
  {
    try
    {
      m_counter++;
      //      pass page to dynamic page management
      IDynamicPageMgmt dpm = m_sessionContext.getDynamicPageMgmt();
    }
  }
}
```

```

        dpm.addDynamicPage(m_layoutXML,"DYNAMIC_EXAMPLE" + m_counter);
        //      switch to dynamic page
        m_subpageURL = dpm.createURLString("DYNAMIC_EXAMPLE" + m_counter);
    }
    catch (Throwable t)
    {
        outputMessage(MT_ERROR,t.toString());
    }
}
}

```

When you choose the **Preview** button, the method `onPreview()` is called. This method takes the text that is entered inside the TEXT control and passes this text to the dynamic page management. The dynamic page management is internally organized on session level - you access it inside an adapter via the member `m_sessionContext`.

Directly inside the method `addDynamicPage(...)`, the XML that is passed gets translated. If the XML contains errors, the corresponding error messages are thrown.

With the message `createURLString(...)`, a URL is built that you can use for referencing the dynamic page. Maybe you want to have a look of what is created as the URL: it is the call of the servlet `StartCISPage` with a parameter indicating the current session and the name of the dynamic page. You should not build this URL on your own but always go through the API.

The URL that is created is used inside the SUBCISPAGE control. If it changes, the control displays the new URL in its content area. This is also the reason why an explicit counter is used: by using the counter, each previewed page gets a URL of its own - the SUBCISPAGE control does not reload its content area if the URL stays stable.

50

Interface IDynamicPageMgmt

The interface looks as follows:

```
package com.softwareag.cis.server;

public interface IDynamicPageMgmt
{
    public class DynamicPageInfo
    {
        String xml;
        String html;
        String[] accessPaths;
    }

    public class URLParameter
    {
        public URLParameter(String name, String value)
        {
            m_name = name;
            m_value = value;
        }
        String m_name;
        String m_value;
    }

    public void addDynamicPage(String xml, String name);
    public void addDynamicPage(String xml,
                               String name,
                               String applicationProject);
    public DynamicPageInfo getDynamicPageInfo(String name);
    public boolean containsDynamicPage(String name);
    public void removeDynamicPage(String name);
    public void clearDynaymicPages();
    public String createURLString(String name, URLParameter[] parameters);
}
```

For more technical details, see the [JavaDoc API documentation](#).

51

Background Information

■ Link to Session Management	210
■ Performance Considerations	211
■ URL Position of the Pages	211
■ Dynamic Pages - Multi Language Management	212

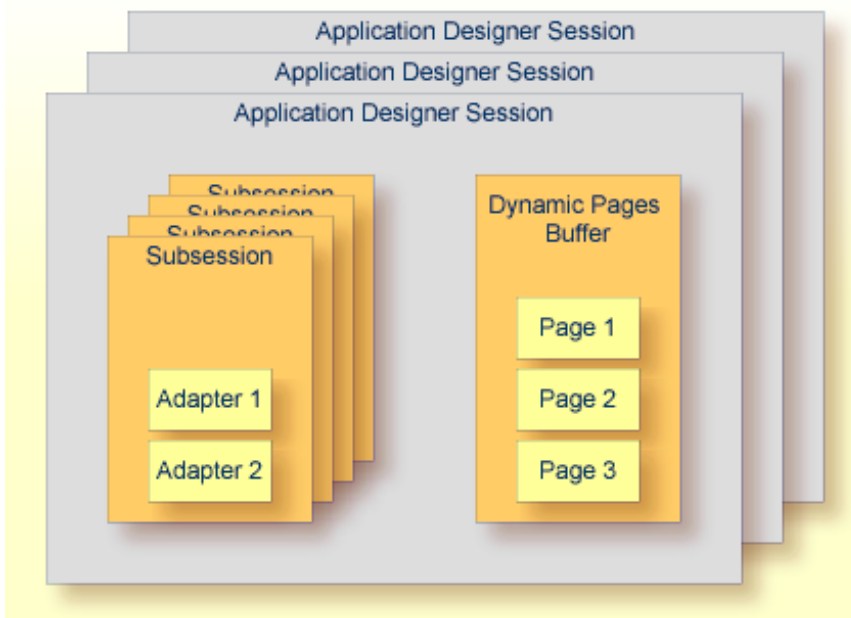
Link to Session Management

The dynamic page management is linked to the session management. Dynamic pages are generated from an XML definition. When calling the method `addDynamicPage(...)` from the interface `IDynamicPageMgmt`, an HTML string is internally generated and kept. It is later picked up by the servlet URL that references the dynamic page.

This means that two aspects are important:

- When is the page taken out of the memory?
- Who can use the page besides the one who has generated it?

Both questions are answered with reference to the session management. See also [Details on Session Management](#).



The above diagram illustrates that a generated page belongs to one session. When the session is removed (e.g. due to log out of the user or due to timeouts), all dynamic pages are released to garbage collection. Of course, you can also remove dynamic pages by using the `removeDynamicPage(...)` method.

Only the session that has created the dynamic page can use it. Parallel sessions are not able to see it; they have to have their own dynamic pages, if required.

Performance Considerations

Dynamic pages are a very flexible technology for building generic application parts. However, this flexibility has some disadvantages when looking at the consumption of resources:

- Normal pages are generated during the design time process; they are already “compiled”. Dynamic pages require an extra generation step during runtime before they can be used.
- Normal pages do not burden the memory because they are stored inside the file system. Dynamic pages are kept in memory. A large page with many controls can be in an area of more than 50-100 kBytes of HTML and JavaScript code. Keep in mind that every user who is logged on holds instances of the pages in the corresponding session context.

Therefore, you should only use dynamic pages when you have specific requirements.

URL Position of the Pages

Normal intelligent pages are located inside a project directory inside the web application that includes Application Designer. Internally, the page is addressed with the following URL:

```
http://<host>:<port>/<webapplication>/<project>/<pagename>.html
```



Note: Remember that you normally do not directly reference pages because they always have to be embedded into a certain environment which is created by the StartCISPage servlet.

If a certain icon is addressed inside the page, the URL of the icon is typically relative to the page's position. Typically, images are kept in a separate directory below the project - e.g. an icon image is positioned inside an *images* directory. In this case, the image is addressed in the following way:

```
images/iconimage.gif
```

Dynamic pages are referenced by the internal usage of a special servlet. The URL that is internally used to access a dynamic page is:

```
http://<host>:<port>/<webapplication>/servlet/StartDynamicPage?SESSIONID=<sessionid>&DYNAMICPAGE=<pageid>
```

This means that from the URL reference point of view, your page is living below the URL root:

```
http://<host>:<port>/<webapplication>/servlet/
```

If you now reference resources which are inside your project's directory, you have to explicitly step into the project. The same icon that was used before, is now referenced via the link:

```
../<project>/iconimage.gif
```

Dynamic Pages - Multi Language Management

For the same reason as explained in the previous section, you must explicitly define the project in which the page is to live when using the multi language management. Multi language files are kept per project; consequently, a page needs to know the project from which it is to take the translated literals.

You define the project by using the following method of `IDynamicPageMgmt` and pass the name of the project.

```
public void addDynamicPage(String xml,  
                           String name,  
                           String applicationProject);
```

You can access the name of the project in which a concrete adapter is living by calling the Adapter method `findPageApplication()`.

52 Dynamic Pages - Dynamic Adapters

If you are using dynamic pages, then you typically also think about dynamic adapters.

Let us take the example of a generic maintenance application for table records in which the detail screen of a record is dynamically created. Having the dynamic screen is the one part of the story; having a generic adapter providing for exactly these properties that are belonging to the records is the other one.

Application Designer offers a powerful way to provide for dynamic adapters. See [*Binding between Page and Adapter*](#) for detailed information.

VIII

Becoming a Member of the Startup Process

There may be the demand to become a member of the startup process of Application Designer: for example, in some cases you have an application which is accessed by Application Designer - by corresponding adapter classes. Typically, you have to initialise this application, for example, by setting up some database connection.

This initialisation takes time and should be done on startup of Application Designer - instead of the first time a user interacts with the application.

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

[Overview](#)

[Startup Class](#)

[Registration](#)

53

Overview

It is quite easy to integrate your application inside Application Designer at startup time. You have to

- provide a startup class supporting the interface
`com.softwareag.cis.server.IServletInitHandler`,
- register this class by editing a configuration file inside Application Designer.

54 Startup Class

The following code shows a simple Java class that can be registered inside the startup process of Application Designer:

```
package com.softwareag.cis.test;

import javax.servlet.*;
import com.softwareag.cis.server.*;

public class StartDemo
    implements IServletInitHandler
{

    public void init(ServletConfig conf)
    {
        System.out.println("StartDemo: started!");
        System.out.println("StartDemo: started!");
        System.out.println("StartDemo: started!");
        System.out.println("StartDemo: started!");
        System.out.println("StartDemo: started!");
    }

    public void destroy()
    {
        System.out.println("StartDemo: destroyed!");
        System.out.println("StartDemo: destroyed!");
        System.out.println("StartDemo: destroyed!");
        System.out.println("StartDemo: destroyed!");
        System.out.println("StartDemo: destroyed!");
    }

}
```

It supports the interface `com.softwareag.cis.server.IServletInitHandler` that requires the implementation of the methods `init` and `destroy`. The `init` method takes the servlet configuration

as parameter with which the Application Designer's servlet itself is initialised. See the documentation of the servlet functions (e.g. in the reference documentation for the servlet API) for more details.

55

Registration

This class must be registered in the */config/statapps.xml* configuration file to be integrated into the startup process of Application Designer. The file looks as follows:

```
<startapps>  
  <start class="com.softwareag.cis.test.StartDemo"/>  
</startapps>
```

Just add a new "start" line and specify the class name. The class must be accessible during runtime.

IX

Adapting the Look & Feel

One of the guiding principles of Application Designer is to provide high-quality controls by simply specifying tags inside a layout definition. Each tag is rendered when generating the intelligent HTML page into various HTML and JavaScript statements. The HTML statements contain the specification of the display style of each control. For example, a label is rendered into a table cell having a defined background (typically a bottom line), a defined text size, etc.

This part describes how to modify the default rendering with the help of style sheets in order to adapt the look and feel to your needs.

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

Introduction

Style Sheet File

Writing a New Style Sheet File

Selecting the Right Style Sheet

Dynamic Selection of the Style Sheet File

Static Selection of the Style Sheet File

56

Introduction

There are different possibilities for adapting the look and feel - depending on what you want to do:

1. Overwrite the style definition in individual controls by specifying the `style` property. Offered for all controls holding text information inside (label, button, field, etc.) and for all container controls (areas, tables, rows, etc.).
2. Exchange the central style sheet file containing all style information for controls. Furthermore, specify your own style sheet: define a style sheet file for a page statically or switch between style sheets dynamically (e.g. user-dependent).
3. Create new controls by yourself and place them into the Application Designer design and runtime environment.

Option 1 is typically used if you like the default style provided by Application Designer - but you want to change it for some pages. For example, you want the text of the button to appear in red - instead of black for some buttons.

Option 2 is typically used if you have to adapt the style of the controls to some customer-specific style. For example, if you want to change the font "Verdana" that is used inside the Application Designer style, or if you want to introduce a new color scheme. Option 2 does not require any changes inside the page layout definitions - the style is completely separated from the layout. You do not have to regenerate your XML definitions at all.

Option 3 is used if you need new controls. There is an open API that allows you to add your own controls in a simple way.

Option 1 is discussed in *Working with Controls* (in the *Layout Elements* documentation). Option 3 is explained in the *Custom Controls* documentation. This part focuses on option 2 - exchanging the style sheet.

57

Style Sheet File

The style information of all controls is defined in the file `<your-webapplication>/cis/styles/CIS_DEFAULT.css`. The style information is sorted alphabetically. Omit the prefix "ROW" or "COL" for container controls - e.g. you find the style information of the "ROWAREA" in "AREA".

```
.AREATable
{
    font-size: 10pt;
    border-width: 0;
    background-color: #E0D8C8;
    border: 1 solid #808080
}
.AREATitleCell
{
    font-size: 8pt;
    color: #808080;
    background-color: #00006C
}
.AREALeftFromTitleCell
{
    font-size: 8pt;
    color: #808080;
    background-color: #00006C
}
.AREARightFromTitleCell
{
    font-size: 8pt;
    color: #808080;
    background-color: #00006C
}
.AREALinks
{
    color: #FFFFFF;
    text-decoration: none
}
```

Take further information out of the comments describing when which style class used.

58

Writing a New Style Sheet File

Style sheet files should be created and maintained with the Style Sheet Editor. This tool covers style sheet manipulation on a very low level. Maintaining style sheets with the Style Sheet Editor means that all information that you enter is kept separate from the style sheet itself.

From release to release, Application Designer adds new controls to its control library. As a consequence, the style sheet template is typically enhanced with every new control. When you work with the Style Sheet Editor, this is done automatically. You just have to regenerate your own style sheet file.

Otherwise (if you have manually created your own style sheet file), you always have to have to embed the enhancements into your style sheet file when Application Designer does style sheet changes: you have to copy the additional Application Designer style classes from the standard Application Designer style sheet file (*CIS_DEFAULT.css*) into your own style sheet file. Use a diff-viewer/diff-editor to do this.

59

Selecting the Right Style Sheet

An intelligent HTML page (generated inside Application Designer) links to a style sheet file. The selection of the style sheet file is done in the following way:

- **Dynamic selection (default):**

The name of the style sheet file is determined by a property `style` of your adapter class. If this is not specified, the default Application Designer style sheet is chosen. The `style` property is provided automatically. See [Dynamic Selection of the Style Sheet File](#) for further information.

- **Static selection:**

The name of the style sheet file is defined in the page by specifying the `stylesheetfile` property of the "page" tag. See [Static Selection of the Style Sheet File](#).

Static selection takes precedence over dynamic selection, i.e. if static selection is defined, dynamic selection is not taken into consideration anymore.

Typically, you define the style sheet file name statically only for certain pages: for those pages you want to be sure that they do not differ from the defined look and feel.

60

Dynamic Selection of the Style Sheet File

■ What You Can Do	234
■ Example	234

The style sheet file is determined by your adapter:

- There is a property `style` with its corresponding `getStyle()` method implemented in the inherited class `com.softwareag.cis.server.Adapter`. The `style` property returns the URL of the used style sheet file.
- The `Adapter` class derives the URL of the style sheet file from the Application Designer session context. Access the Application Designer session context by the protected property `m_sessionContext`. The `m_sessionContext` object provides a `setStyle()` and `getStyle()` method. To change the style sheet file inside the adapter, do the following:

```
public void ...()
{
    ...
    m_sessionContext.setStyle("...yourStyleURL... ");
    ...
}
```

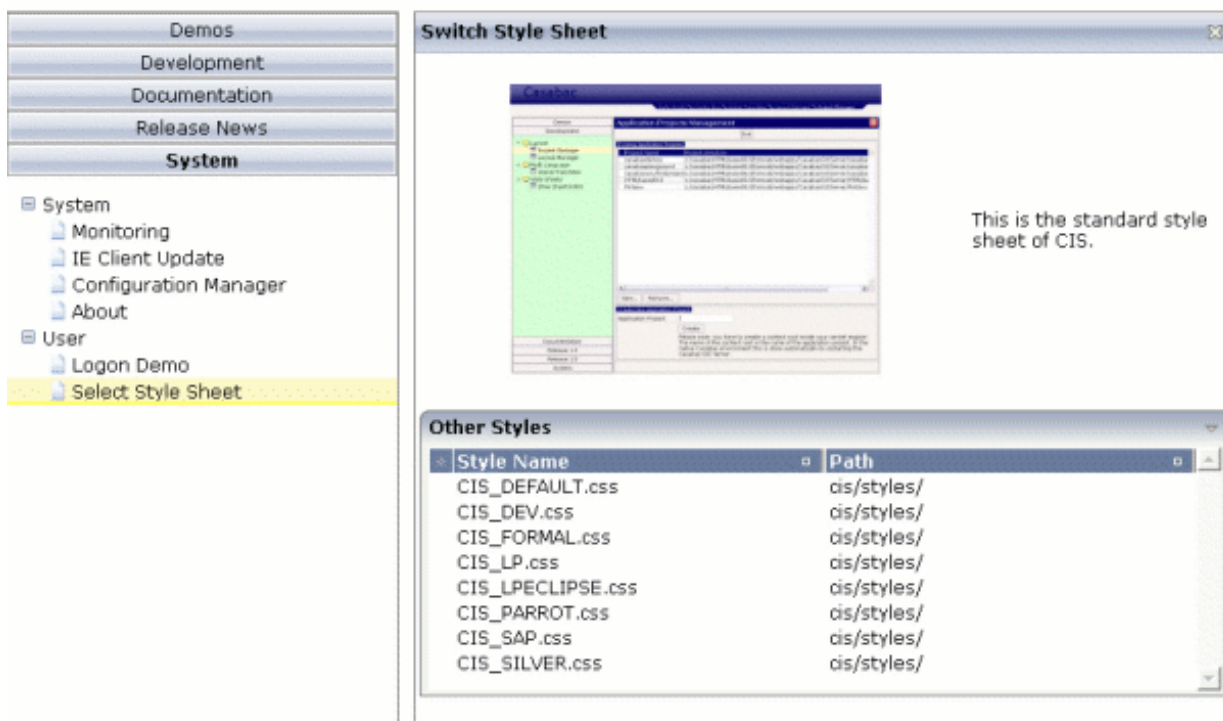
What You Can Do

There are two options that you can use in parallel:

- You can take over the `getStyle()` method in your adapter from the `Adapter` class. In this case, you can set the session's style sheet via `m_sessionContext.setStyle(...)`, as described.
- You can write your own `getStyle()` method and can apply any other rule you might think of on your own.

Example

Inside the Application Designer demo workplace, there is a function to select a style sheet for your current session:



The program lists all available style sheets in the directory `<webapp>/styles/`. If you select one style sheet file, then the selected style sheet is internally passed to the session context as described in the previous section.

Consequently, all pages in the content area of the workplace will be rendered with this style sheet.

The style of the workplace itself will not change: the workplace adapter overwrites the `getStyle()` method: with the workplace, you can pass its style sheet file when dynamically defining the workplace.

61

Static Selection of the Style Sheet File

It makes sense for some pages to define the style sheet file statically. In this case, it cannot be changed dynamically. This can be done inside the XML layout definition of the page with the "page" tag.

```
<page model="xyz"
      pagename="xyz.html"
      stylesheetfile="/HTMLBasedGUI/general/layout.css">
    ...
    ...
    ...
</page>
```


X

Controls for Database Reporting

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

Basics

DBQUERY

DBFIELD

DBCOMBO

DBSELECTOPTION

DBCHECKBOX

DBRADIOBUTTON

62

Basics

■ Two Types of DB Controls	242
■ When to Use Which Type	243

Two Types of DB Controls

Application Designer provides a simple but flexible way to develop typical reporting pages for querying the contents of a relational database. There is a set of database (DB) controls for creating queries. DB controls can be divided into two types:

- **Filter Criteria Controls (DBSELECTOPTION, DBFIELD, DBCOMBO, DBRADIOBUTTON and DBCHECKBOX)**

These are controls representing/covering a single filter criterion of a query. The controls provide for value help (except DBCHECKBOX and DBRADIOBUTTON), a check whether the user input is valid (again except DBCHECKBOX and DBRADIOBUTTON) and a conversion of the user input to a string to be added to the SQL string. The following screen shot shows (from top to bottom) a DBCHECKBOX, DBCOMBO, DBFIELD, DBRADIOBUTTON and DBSELECTOPTION control.



The screenshot shows a window titled "Controls Demo" with a light blue header and a grey border. Inside, there are five rows of controls:

Color Printer	<input checked="" type="checkbox"/>
Location	Office 102 
Manufacturer	
Name	Brother HL-1030 Series  HP LaserJet 5MP 
Resolution	600 

As with the "corresponding" controls (CHECKBOX, COMBODYN, FIELD, RADIOBUTTON) you have a high degree of freedom when placing the DB controls into the layout of the report.

- **DBQUERY Control**

This control represents an entire database query. It covers filter criteria, the query execution, the result area, output formats such as PDF and query variants - with a minimum of programming effort on server side. The following screenshot shows one DBQUERY control.

Report Demo

Address Report

Title

First Name

Last Name

Line 0 - 7 / 11

* Title	Last Name	First Name	Street	Cou	St
Dr	Ahlenfeld	Susanne	Westring 3	DE	No
		Susanne	Grossfelderstr. 3	DE	No
Mr	Duck	Dagobert	Grossfelderstr. 3	DE	No
		Donald	Grossfelderstr. 3	DE	No
MSS	Hahn	Ulla	Poststrasse	DE	
Mr	Max	Martin	Bahnhofstr 7	DE	
	Schumacher	Michael	Weingartenstr 42	DE	

CSV

The DBQUERY control has various properties in order to adapt the layout to your needs - but you are not as free as with the above mentioned DB controls.

When to Use Which Type

Use the filter criteria controls in the following case:

- The page layout of the report is of high importance and the DBQUERY control cannot be adapted to match your needs.

Use the DBQUERY control in the following cases:

- If you want to create a report in a very efficient way (in respect to creating the page layout as well as to the programming effort on the server side).
- If you do not want to care about report variants and PDF conversion

63

DBQUERY

■ Example	247
■ DBQUERY Properties	251
■ DBFILTER Properties	255
■ DBCOLUMN Properties	257
■ DBPARAMSINGLEVALUE Properties	259
■ DBPARAMDOUBLEVALUE Properties	260
■ Variant Management	261
■ PDF Generation	262

The DBQUERY control is designed to significantly reduce the effort for developing queries against relational databases. In the control definition, you specify the SELECT string, the filter criteria and the output columns - the rest is done automatically. The sequence, sorting and grouping of the output grid can be defined in a very flexible way. You even can store so-called variants: if you have certain filter criteria you always want to use in a query, then you can store them under a name in order to have quick access to often-used queries. The following typical aspects of a query are covered:

- Filter criteria (ad hoc input, saved in query variants).
- Result output (server-side scrolling, context menu).
- Report variants (column visibility and column order, sorting, grouping, etc.).
- (Default) PDF generation.

Example

Address Report Lastname A to T

Title

Last Name A>>T

Execute

Title	Last Name	First Name	Street	Cour	Sta
Dr	Ahlenfeld	Susanne	Westring 3	DE	No
		Susanne	Grossfelderstr. 3	DE	No
Mr	Duck	Dagobert	Grossfelderstr. 3	DE	No
		Donald	Grossfelderstr. 3	DE	No
MSS	Hahn	Ulla	Poststrasse	DE	
Mr	Max	Martin	Bahnhofstr 7	DE	
	Schumacher	Michael	Weingartenstr 42	DE	
		Michael	Hudson Drive	US	Ne
		Michael	Ocean View	US	Ca
		Ralf	Broad Way	US	Ne

Variant Management Line 0 - 10 / 10

Output Formats

There is a DBQUERY control with its inner components. Look at the corresponding layout definition:

```
<dbquery valueprop="dbQueryInfo" query="SELECT * FROM ADDRESS AS A, BUSINESSPARTNER AS B
WHERE A.BUSINESSPARTNERID = B.ID" datasource="addressdb" title="Address Report"
rowareaname="Report Demo" executebuttonname="Execute" maxrows="200"
image="images/addresses.gif" height="100%" rowcount="40" titletext="Address Report">
  <dbfilter labelname="Title" labelwidth="150" querycolumn="title" valuehelptable="title"
valuehelpcolumn="id" fieldwidth="200" hideout="true">
  </dbfilter>
  <dbfilter labelname="First Name" labelwidth="150" querycolumn="firstname" fieldwidth="200">
  </dbfilter>
  <dbfilter labelname="Lat Name" labelwidth="150" querycolumn="lastname" fieldwidth="200">
  </dbfilter>
  <dbcolumn name="Title" column="TITLE" width="50" widthpdf="2cm" groupby="true">
  </dbcolumn>
  <dbcolumn name="Last Name" column="LASTNAME" width="100" widthpdf="3cm">
```

```
sortorder="1"
    sortascending="true" groupby="true">
  </dbcolumn>
  <dbcolumn name="First Name" column="FIRSTNAME" width="100" widthpdf="3cm" ↵
sortorder="2"
    sortascending="true">
  </dbcolumn>
  <dbcolumn name="Street" column="STREET" width="150" widthpdf="5cm">
  </dbcolumn>
  <dbcolumn name="Country" column="COUNTRY" width="50" widthpdf="2cm">
  </dbcolumn>
  <dbcolumn name="State" column="STATE" width="100%" widthpdf="5cm">
  </dbcolumn>
</dbquery>
```

Look at the following items:

- There is a DBQUERY definition with the property valueprop "dbQueryInfo". The property query contains the SELECT string.
- There are DBFILTER definitions for each filter criterion.
- There are DBCOLUMN definitions for each grid column of the result area.

The adapter code is the following:

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

import java.sql.Connection;
import java.sql.DriverManager;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.util.DBQUERYDataObject;
import com.softwareag.cis.server.util.DBQUERYInfo;
import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBTEXTGRIDLine;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;
import com.softwareag.cis.server.util.IDBQUERYContextMenuRequestListener;
import com.softwareag.cis.server.util.IDBQUERYGeneratePDFRequestListener;
import com.softwareag.cis.server.util.IDBQUERYOptimizer;
import com.softwareag.cis.server.util.MENUNODEInfo;
import com.softwareag.cis.server.util.TREECollection;
import com.softwareag.cis.server.util.ValidValueLine;

public class DBQUERYAdapter
    extends Adapter
    implements IDBDemoAdapter
{
    // -----
    // inner classes
```

```

// -----

/** class used for a simple connection management. */
public class ConnectionProvider
    implements IDBQUERYConnectionProvider
{
    public Connection getDBConnection(String datasource)
    {
        try
        {
            Class.forName("org.hsqldb.jdbcDriver");
            return DriverManager.getConnection("jdbc:hsqldb:hsq://localhost", "sa", "");
        }
        catch (Exception exc)
        {
            throw new DelegateError(exc);
        }
    }
}

/** class used for SQL optimization. */
public class SQLOptimizer
    implements IDBQUERYOptimizer
{
    public String optimizeQuery(String query, String[] gridColumns)
    {
        // do checks + optimization here
        return query;
    }
}

/** class used for pop-up menu. */
public class MyMenuNodeInfo
    extends MENUNODEInfo
{
    IDynamicAccess m_row; // = row for that the context menu is requested
    public MyMenuNodeInfo(IDynamicAccess row, String text, String image)
    {
        super(text, image);
        m_row = row;
    }
    public void reactOnSelect()
    {
        outputMessage("S", getText() + " selected");
    }
}

/** class used to create the context menu within the result grid. */
public class ResultAreaContextMenu
    implements IDBQUERYContextMenuRequestListener
{
    public void reactOnContextMenuRequestFor(IDynamicAccess row)
    {
        // opens a pop-up menu with two entries
    }
}

```

```
        TREECollection cm = new TREECollection();
        cm.addTopNode(new MyMenuNodeInfo(row, "Edit", "images/edit.gif"),true);
        cm.addTopNode(new MyMenuNodeInfo(row,"Remove", "images/remove.gif"),true);
        showPopupMenu(cm);
    }
}
/** class used for PDF conversion. */
public class GeneratePDFRequestListener
implements IDBQUERYGeneratePDFRequestListener
{
    public void generatePDFAndDisplayDocument(DBQUERYDataObject dataObject)
    {
        // create PDF document here
    }
}

DBQUERYInfo m_dbQueryInfo = new DBQUERYInfo(this,new ConnectionProvider(),
        new ResultAreaContextMenu(),
        new SQLOptimizer(),
        new GeneratePDFRequestListener());
public DBQUERYInfo getDBQueryInfo() { return m_dbQueryInfo; }
}
```

Programming the DBQUERY is quite simple. Define an instance of the class `DBQUERYInfo`. This instance is referenced by the `valueprop` definition inside the DBQUERY tag. The `DBQUERYInfo` offers a set of constructors.

Mandatory Parameters

Same to all constructors are the (two) mandatory parameters. First you have to pass the adapter object that defines the `DBQUERYInfo` instance. This object is used, for example, to open a pop-up inside the DBQUERY control. With the second, you have to pass an implementation of interface `IDBQUERYConnectionProvider`. As the `DBQUERYInfo` class does not open a database connection on its own, it uses this object to obtain a connection. The connection is only used to read data from the database. There are no updates (insert/update/delete) done with this connection. Internally, the provided connection is buffered and used each time the query is executed. As the `DBQUERYInfo` does not open the connection, it does not care about closing the connection.

Optional Parameters

The constructors vary in the list of their optional parameters. By construction, you may pass an implementation of interface `IDBQUERYContextMenuRequestListener`. This object is called (with method `reactOnContextMenuRequestFor`) if the user clicks into a line inside of the report result grid with right mouse button. See the Javadoc documentation of interface `IDBQUERYContextMenuRequestListener` for details.

If you want to check/optimize the SQL statement prior its execution, you may pass an implementation of interface `IDBQUERYOptimizer`. The object is called with method `optimizeQuery` each time the report is executed. See the Javadoc documentation of interface `IDBQUERYOptimizer` for details.

If you do not want to use the default PDF conversion of the DBQUERY control, you may pass an implementation of interface `IDBQUERYGeneratePDFRequestListener`. This object is called (with method `generateAndDisplayPDF`) if the user clicks the "PDF" icon within the DBQUERY control. See the Javadoc documentation of interface `IDBQUERYGeneratePDFRequestListener` for details.

DBQUERY Properties

Java Binding			
valueprop	Property VALUEPROP points to a property of type <code>IDBQUERYInfo</code> (package <code>com.softwareag.cis.server.util</code>). This class encapsulates the reports execution, the variant management and the PDF and CSV output.	Obligatory	
directselectmethod	The <code>DIRECTSELECTMETHOD</code> property is used to point to a method of your adapter class, which is called when a selection event occurs within the result grid.	Optional	
directselectevent	The <code>DIRECTSELECTEVENT</code> property is used to define whether the direct select method is called by a single or a double click. Typically you use a single click ("onclick") if you want to select something in the grid and to display simultaneously details of what was selected in the same page. Use a double click ("ondblclick") to navigate to the next page.	Optional	ondblclick onclick
DB Binding			
datasource	Logical identifier of the database on that the report is executed. This value is passed in the method <code>"IDBQUERYConnectionProvider.GetDBConnection"</code> .	Obligatory	
query	SQL statement with a complete <code>SELECT</code> and <code>FROM</code> clause and with an optional <code>WHERE</code> clause. With the <code>SELECT</code> clause you define the result set of the report (each <code>DBCOLUMN</code> control refers to one element/column name of this result set via the property <code>COLUMN</code>). Prior the reports execution the values of the <code>DBFILTER</code> controls are added to this query.	Obligatory	

maxrows	Specifies the maximum number of rows fetched from database. The value "0" represents unlimited. Default is "200".	Optional	20 50 100 200 500 0
executeonload	Flag which indicates if the report is to be executed on page load. Default is "false".	Optional	true false
filterlinkoperator	The values of the DBFILTER controls are added dynamically to QUERY prior the reports execution. With this property you can specify the operator to be used to add the DBFILTER values. Default is "AND".	Optional	and or
Height			
height	The height of the DBQUERY control in pixels or as percentage value.	Obligatory	100 150 200 250 300 250 400 50% 100%
Title			
title	Name of the database report.	Optional	
titletextid	Text ID (report title) for the multi language management.	Optional	
titlelabelwidth	Width of the title in pixels or as percentage value.	Optional	100 120 140 160

			180 200 50% 100%
titlestyle	Direct manipulation of title style.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
Row Area			
rowareaname	Name of the surrounding row area.	Optional	
rowareatextid	Text ID (row area) for the multi language management.	Optional	
foldable	The surrounding row area can be shrinked by clicking on its title. This standard behaviour can be disabled by setting FOLDABLE to "false".	Optional	true false
rowareastyle	Inline style for the surrounding row area.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
image	URL of the image that is shown at the right hand of the filters. The URL can be relative or absolute.	Optional	
executebuttonname	Name that is displayed on the execute button.	Optional	
executebuttontextid	Text ID (execute button) for the multi language management.	Optional	
outputmesstostatusbar	Flag that indicates if messages that are generated by the report (DBQUERYInfo) are displayed within the status bar (default) or inside the DBQUERY control above the result grid ("false").	Optional	true false
showprintversion	If switched to "true" then a small print icon will appear right from the grid. The print icon opens up a modal popup from which the HTML produced inside the report can be directly sent to the printer. Pay attention: if specifying "true" then the adapter property holding the REPORTInfo object must create the REPORTInfo instance with passing "this" in the constructor.	Optional	true false

showpdf	<p>If set to "true" then a PDF icon is rendered in the right top corner of the control. When the user clicks on the icon then the report is automatically rendered as pdf - and the result will show up in a popup window.</p> <p>Pay attention: if setting this property to "true" then you also have to choose a special constructor when creating the REPORTInfo instance on server side, in which the instance of the model is passed as argument.</p>	Optional	true false
personalizable	<p>If defined to "false" then no re-arranging of columns is offered to the user.</p> <p>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.</p>	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Result Grid			
rowcount	The property ROWCOUNT defines the number of rows that are fetched from the server.	Optional	1 2 3 int-value
withselectioncolumn	Flag which indicates if the result grid does have a selection column. Default is "true".	Optional	true false
withtitlerow	Flag indicates if the result grid does have a title row (default) or not ("false").	Optional	true false
hscroll	Indicator if the result grid shows a horizontal scroll bar.	Optional	true false
fixlayout	When switching the FIXLAYOUT property to value "true" then internally the result grid is arranged in a way that the area always determines its size out of the width specification of the DBCOLUMN controls. The browser does not look into the column contents in order to try to optimise the size of the	Optional	true false

	area - but always follows the width and height that you define.		
backgroundstyle	Direct style manipulation of the table style which surrounds the table cells inside the result grid.	Optional	

DBFILTER Properties

The DBFILTER tag is the typical tag that is placed inside a DBQUERY definition. The DBFILTER defines one filter criterion with its binding to a table column (property `querycolumn`). The parameter values are added dynamically to the WHERE clause of the SQL query prior to report execution. The SQL statement is defined within the `query` property of the DBQUERY control.

DB Binding			
querycolumn	Name of the column within the reports query the DBFILTER control is bound to. With this name user input will be added to the SQL statement on report execution. The SQL statement is defined in the property QUERY of the DBQUERY control. Example: If you define the SQL statement like "SELECT STREET, ZIPCODE, TOWN FROM ADDRESS" in property QUERY of the DBQUERY control you can specify any column name of table "ADDRESS" here.	Obligatory	
valuehelptable	You may have a table in that the valid values for this filter are kept. In that case you can provide for a filter value help by using the properties VALUEHELPTABLE and VALUEHELPCOLUMN. Input the name of the table here.	Optional	
valuehelpcolumn	You may have a table in that the valid values for this filter are kept. In that case you can provide for a filter value help by using the properties VALUEHELPTABLE and VALUEHELPCOLUMN. Input the name of the column here.	Optional	
usequeryforvaluehelp	Flag that indicates that the value help is coming from the query result set.	Optional	true false
columnalias	Alias of the database column	Optional	
Filter Name			
labelname	Name of the filter.	Optional	
labeltextid	Text ID (filter name) for the multi language management.	Optional	
labelwidth	Width of the filter name in pixels or as percentage value.	Optional	100 120 140

			160 180 200 50% 100%
Filter Input			
fieldwidth	Width of the filter input field in pixels or as percentage value.	Optional	100 120 140 160 180 200 50% 100%
fieldlength	Width of the filter input field in amount of characters. FIELDWIDTH and FIELDLENGTH should not be used together.	Optional	5 10 15 20 int-value
felddatatype	Specifies the data type of the filter. As a consequence the fields inside the grid of the value help popup are checking the data during input (e.g. if the DATATYPE is "int", it is not allowed to enter alphabetic characters) and adds a logic to transfer the data into various output formats (e.g. if the DATATYPE is "date", the date is formatted into the right date format). In addition these fields have a standard "value help" popup dialog for some data types (e.g. if the DATATYPE is "date" then automatically a date input dialog pops up if invoking "value help").	Optional	int float date
fielduppercase	Flag which indicates if the alphabetic characters input should be converted to upper case if necessary. Default is "false".	Optional	true false

fieldstyle	Explicit style information for the input field. Example: if you want the text to be right aligned, define "text-align: right".	Optional	
fieldhelpid	Name that is used to identify the online help page to be opened if the user presses the F1-key inside the FIELD control. Please refer to chapter "Online Help Management" for details.	Optional	
Comment			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

DBCOLUMN Properties

The DBCOLUMN tag is the typical tag that is placed inside a DBQUERY definition. The DBCOLUMN definition defines one column within the result grid. It is bound to a name (column name or column alias) in the result set of the report. This result set is defined by the `query` property inside the DBQUERY definition.

DB Binding			
column	Name of the table column the DBCOLUMN control refers to. Please pay attention: If you use column aliases in the SQL query (refer to property QUERY of the DBQUERY control) you must specify the column alias here. Example: If you use the query "Select CUSTNAME as CN FROM CUSTOMER" you have to use column alias "CN" here. "	Obligatory	
Appearance			
name	Name of the title cell of the grid column.	Optional	
textid	Text ID (name) for the multi language management.	Optional	
width	Width of the column in pixels or as percentage value.	Optional	100 120 140 160 180 200 50% 100%
widthpdf	Width in centimetres the column should occupy inside the PDF document.	Optional	0.5 0.75

			1 2 5 10
align	Horizontal alignment of the text within the column. Default is "left", other values are "center" or "right".	Optional	left center right
straighttext	Flag which indicates whether the text displayed inside the column is formatted as HTML text or as straight text. Default is "false".	Optional	true false
convertspaces	Flag which indicates if spaces inside the text of a cell should be converted in "non-breakable-spaces". In general HTML converts several appearances of space-characters ("blanks") into one space-character. If you set CONVERTSPACES to "true", this default behaviour is switched off.	Optional	true false
cuttextline	If a text does not fit into a cell then it is cut off. If you set CUTTEXTLINE to "false", it will be broken - following HTML rules for breaking text. Therefore the cell will contain more than one text line.	Optional	true false
datatype	Data type of the content of the column. Therefore certain rendering rules are applied (e.g. in case of "date", a YYYYMMDD date is converted into a proper date format).	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
Comment			
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

DBPARAMSINGLEVALUE Properties

The DBPARAMSINGLEVALUE tag is the typical tag that is placed inside a DBQUERY definition. The DBPARAMSINGLEVALUE defines a filter criterion. In contrast to the DBFILTER tag, it is not visualized within the DBQUERY control and is therefore not accessible/changeable by the user.

Example: a table "ADDRESS" may have a column "COUNTRY". You want to restrict the user to see German addresses only. In this case, use a DBPARAMSINGLEVALUE with the property `querycolumn` set to "COUNTRY", operator set to "=" (equal), and `value` set to "DE".

Basic			
querycolumn	Name of the column the DBPARAMSINGLEVALUE control refers to. This name is used to add the parameters value to the SQL statement prior the reports execution. The SQL statement is defined in the property QUERY of the DBQUERY control.	Obligatory	
operator	Name of the operator to use to append the value to WHERE clause of the SQL statement.	Obligatory	= != ~ !~ > <= <

			<= NULL NOT NULL
value	The parameter value. You can either enter a fixed value or you can specify the name with that a value can be looked up from the session context when executing the report.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

DBPARAMDOUBLEVALUE Properties

The DBPARAMDOUBLEVALUE tag is very similar to DBPARAMSINGLEVALUE. The only difference is that you can specify operators that work on two operands (e.g. "is between").

Basic			
querycolumn	Name of the column the DBPARAMSINGLEVALUE control refers to. This name is used to add the parameters value to the SQL statement prior the reports execution. The SQL statement is defined in the property QUERY of the DBQUERY control. Example: If you define the SQL statement like "SELECT STREET, ZIPCODE, TOWN FROM ADDRESS" in property QUERY of the DBQUERY control you can specify any column name of table "ADDRESS" here.	Obligatory	
operator	Name of the operator to use to append the value to WHERE clause of the SQL statement. The operator specified here must work on two operands. At the moment "between" and "not between" are supported.	Obligatory	>> !gt;>
value1	The first parameter value. You can either enter a fixed value or you can specify the name with that a value can be looked up from the session context when executing the report. Example: If you enter "\$FROMDATE\$" here, the DBPARAMSINGLEVALUE will try to lookup a value bound to the name "FROMDATE" from the session context. An exception will be raised if nothing is found.	Obligatory	
value2	The second parameter value.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Variant Management

The DBQUERY control provides for so-called query variants. Within a variant, you can save e.g. an often-used filter criteria combination. If you do not want to use variants at all, you just set the property `personalizable` to `false`.

Persistence

The variant data is stored in the local file system.

Key

The key of a report variant is consists of:

- the name of the web application,
- the name of the project that contains the HTML page,
- the name of the HTML page that contains the DBQUERY control,
- the actual user logged on (read from the session context on save),
- the name of the variant (specified in the save pop-up).

As the name of the page (that embeds the DBQUERY control) is part of the variant key, you can have multiple pages within one Application Designer project - each with its own variant set. But: as a consequence, variants do not work inside the Layout Painter. Reason: if you preview the XML layout definition, the editor first removes the temporary HTML page (name starts with "ZZZZZZZZ") of a former preview. In a second step, it creates a new page - with a different name (the name contains a timestamp). This means: inside the Layout Painter, you never view the same HTML page twice. If you work with the published HTML page, variants will work properly.

Create/Change/Remove

To create a variant, you just save your current input (filter criteria or settings inside the variant properties pop-up). To save a variant, choose the save icon to the right of the result grid. To change an existing variant, open the variant (with input field to the right of the report title label), apply your changes (either by changing the filter criteria or within the variant properties pop-up - icon above the save icon) and save them. To remove a variant, choose the remove icon (icon below save icon), select one or more variants and choose ok.

PDF Generation

The DBQUERY control provides for a generation of a PDF document. The generation is invoked when choosing the "PDF" icon to the right of the result grid. The document is displayed inside a pop-up. It contains the following data:

- Report title (header line).
- Timestamp of the reports execution (footer).
- Table with used filter criteria (body).
- Table with result of the report (body).

How to do the PDF generation on your own?

The `DBQUERYInfo` class provides for a constructor where you can pass an implementation of interface `IDBQUERYGeneratePDFRequestListener`. This implementation is called with the method `generatePDFAndDisplayDocument` if the "PDF" icon is chosen. The current data of the DBQUERY is passed within that call. For details, see the JavaDoc documentation of interface `IDBQUERYGeneratePDFRequestListener` and class `DBQUERYInfo` (both from package `com.softwareag.cis.server.util`).

64 DBFIELD

▪ Example	264
▪ Properties	269

The DBFIELD control represents a filter criterion of a database query. It provides for a value help that is read from the database, a convenient way to append the filter criterion to the SELECT statement, and the ability to reflect a “to-one” dependency between filter criteria.

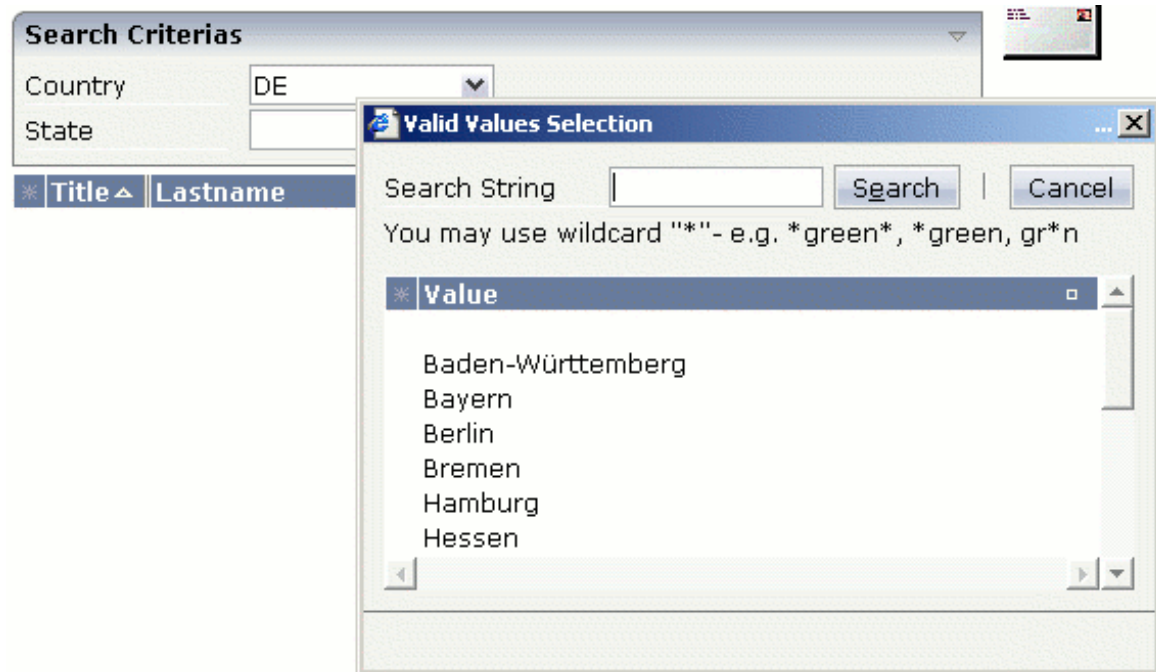
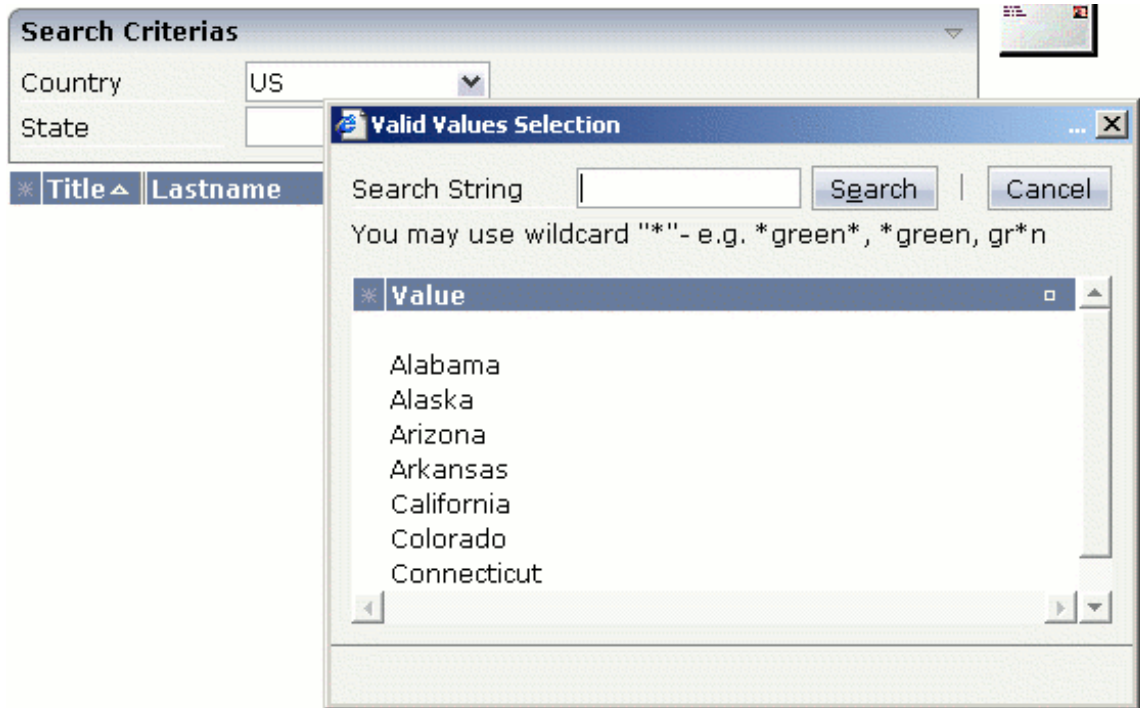
Example

The following image shows an example in which two DBFIELD controls are used for the filter criteria "Country" and "State" within a simple business partner report. Both fields provide for value help. The field "State" is defined to be dependent on "Country". As a consequence, the list of valid values for "State" is country-specific. The result shows partners that reside in state "New York" of the United States of America.

The screenshot displays a user interface for a business partner report. It features a 'Filter Criteria' section with two dropdown menus: 'Country' set to 'US' and 'State' set to 'New York'. An 'Execute' button is located to the right of the 'State' dropdown. Below this is a 'Result' section containing a table with the following data:

Title	Last Name	First Name	Cou	State
Mr	Schumacher	Michael	US	New York
Mr	Schumacher	Ralf	US	New York

The following screenshots demonstrate the dependency between country and state. The first pop-up shows the valid states if country is set to "US". The second pop-up shows the valid states in Germany.



Have a look at the XML layout definition:

```
<rowarea name="Search Criteria">
  <itr>
    <label name="Country" width="60">
      </label>
      <dbfield valueprop="dBFieldCountry" querycolumn="COUNTRY" ↵
datasource="addressdb"
                valuehelptable="COUNTRY" valuehelpcolumn="ID">
      </dbfield>
    </itr>
    <itr>
      <label name="State" width="60">
        </label>
        <dbfield valueprop="dBFieldState" querycolumn="STATE" datasource="addressdb"
valuehelpcolumncond="COUNTRY">
                valuehelptable="STATE" valuehelpcolumn="ID" ↵
      </dbfield>
    </itr>
    <hdist width="100%">
      </hdist>
      <button name="Execute" method="onExecute">
      </button>
    </itr>
  </rowarea>
<rowarea name="Result" height="140">
  <itr height="100%">
    <textgridsss2 griddataprop="result" rowcount="5" width="100%">
      <column name="Title" property="TITLE" width="50">
      </column>
      <column name="Last Name" property="LASTNAME" width="100">
      </column>
      <column name="First Name" property="FIRSTNAME" width="100">
      </column>
      <column name="Country" property="COUNTRY" width="50">
      </column>
      <column name="State" property="STATE" width="100%">
      </column>
    </textgridsss2>
  </itr>
</rowarea>
```

The corresponding adapter code is:

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

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.util.DBFIELDInfo;
import com.softwareag.cis.server.util.DBQUERYDataObject;
import com.softwareag.cis.server.util.DBQUERYInfo;
```

```

import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBTEXTGRIDLine;
import com.softwareag.cis.server.util.DBUtil;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBCondition;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;
import com.softwareag.cis.server.util.IDBQUERYContextMenuRequestListener;
import com.softwareag.cis.server.util.IDBQUERYGeneratePDFRequestListener;
import com.softwareag.cis.server.util.IDBQUERYOptimizer;
import com.softwareag.cis.server.util.MENUNODEInfo;
import com.softwareag.cis.server.util.TREECollection;

public class DBFIELD_Adapter
    extends Adapter
    implements IDBQUERYConnectionProvider, IDBDemoAdapter
{
    private Connection m_connection;

    // property >DBFieldCountry<
    DBFIELDInfo m_dBFieldCountry = new DBFIELDInfo(this);
    public DBFIELDInfo getDBFieldCountry() { return m_dBFieldCountry; }

    // property >DBFieldState<
    DBFIELDInfo m_dBFieldState = new DBFIELDInfo(this, m_dBFieldCountry);
    public DBFIELDInfo getDBFieldState() { return m_dBFieldState; }

    // property >result<
    DBTEXTGRIDCollection m_result = new DBTEXTGRIDCollection();
    public DBTEXTGRIDCollection getResult() { return m_result; }
    public void setResult(DBTEXTGRIDCollection value) { m_result = value; }
    ...
    // -----
    // inner classes
    // -----
    /** class used for a simple connection management. */
    public class ConnectionProvider
        implements IDBQUERYConnectionProvider
    {
        public Connection getDBConnection(String datasource)
        {
            try
            {
                Class.forName("org.hsqldb.jdbcDriver");
                return DriverManager.getConnection("jdbc:hsqldb:hsqldb://localhost", ↵
"sa", "");
            }
            catch (Exception exc)
            {
                throw new DelegateError(exc);
            }
        }
    }
}

```

```
...
    public void onExecute()
    {
        try
        {
            StringBuffer sb = new StringBuffer();
            sb.append("SELECT * FROM BUSINESSPARTNER INNER JOIN ADDRESS ON ↵
BUSINESSPARTNER.ID =
                                ADDRESS.BUSINESSPARTNERID");
            DBUtil.addToQuery(sb, new IDBCondition[] { m_dBFieldCountry, ↵
m_dBFieldState}, true);
            String dataSource = m_dBFieldCountry.getDataSource();
            Connection con = getDBConnection(dataSource);
            ResultSet rs = con.createStatement().executeQuery(sb.toString());
            m_result.initWithResultSet(rs);
        }
        catch (Exception exc)
        {
            throw new DelegateError(exc);
        }
    }
...
}
```

Both properties `dBFieldCountry` and `dBFieldState` are of type `DBFIELDInfo` (from the package `com.softwareag.cis.server.util`). The `DBFIELDInfo` implements (like all DB controls) the interface `IDBCondition` (`com.softwareag.cis.server.util`). With class `DBUtil` (`com.softwareag.cis.server.util`), you can append the values of the filter criteria to the `SELECT` statement in a convenient way. See the JavaDoc documentation of class `DBUtil` for details.

The `DBFIELDInfo` class does not open a database connection on its own (same to all DB controls). The embedding adapter provides for an implementation of interface `IDBQUERYConnectionProvider` (`com.softwareag.cis.server.util`) when creating a `DBFIELDInfo` object. The interface method `getDBConnection` is called once - at the first time the `DBFIELDInfo` accesses the database. There are no updates (insert/update/delete) done with this connection. As the `DBFIELDInfo` does not open the connection, it does not care about closing the connection.

You see that the object `DBFIELDInfo dBFieldCountry` is passed in the constructor of `DBFIELDInfo dBFieldState`. With this, you define `DBFIELDInfo dBFieldState` depending on `dBFieldCountry`. The list of valid states only shows items that belongs to the country actually set.

For displaying the result, the class `DBTEXTGRIDCollection` (from the package `com.softwareag.cis.server.util`) is used. This class extends `TEXTGRIDCollection` by the ability to initialise the collection with a result set (`java.sql.ResultSet`). For each line of the result set, it creates an object of class `DBTEXTGRIDLine` (package `com.softwareag.cis.server.util`). Class `DBTEXTGRIDLine` implements the interface `IDynamicAccess`. With this, "normal" text grid controls can be used to visualize the data of the `DBTEXTGRIDCollection`.

Properties

Basic			
valueprop	Property that returns a DBFIELDInfo-instance. This instance provides for the value help read from database as well as for a convenient way to append the filter value to query string.	Obligatory	
querycolumn	Name of the column in the query to that the filter criteria is belongs to. This column may differ from the value help table/column (properties VALUEHELPTABLE, VALUEHELPCOLUMN). This name is used to build a SQL string in method "toSQLString".	Obligatory	
datasource	Logical identifier of the data source to use. This name is passed to the connection provider in method "IDBQUERYConnectionProvider.getDBConnection".	Obligatory	
valuehelptable	Name of the table from there the list of valid values can be read.	Obligatory	
valuehelpcolumn	Name of the column from there the list of valid values can be read.	Optional	
valuehelpcolumndescr	Name of a column where an additional description of the valid values is stored. The name must identify a column inside the "value help table" (property VALUEHELPTABLE).	Optional	
valuehelpcolumncond	Name of the column inside the "value help table" (property VALUEHELPTABLE) that defines the "to-one" dependency to another DBFIELD control.	Optional	
width	Width of DBFIELD in pixels or as percentage value.	Optional	
length	Width of DBFIELD in amount of characters. WIDTH and LENGTH should not be used together.	Optional	
datatype	By default, the DBFIELD is managing its content as a string. By the DATATYPE property, force the type of the data that is represented. As a consequence the DBFIELD is checking the data during input (e.g. if the DATATYPE is "int", it is not allowed to enter alphabetic characters) and adds a logic to transfer the data into various output formats (e.g. if the DATATYPE is "date", the date is formatted into the right date format).	Optional	int float date
flush	Flushing behaviour, please view "Common Rules" for details	Optional	screen server
displayonly	If set to "true", the DBFIELD will not be accessible for input. It is just used as an output field.	Optional	true false
align	Explicit Alignment	Optional	

valign	Explicit Alignment	Optional	
colspan	Number of columns occupied by this control.	Optional	
rowspan	Number of rows occupied by this control.	Optional	
fieldstyle	Explicit style information passed to the DBFIELD. Example: if you want the text inside the DBFIELD to be right aligned, define "text-align: right".	Optional	
helpid	Identifier that is used for building the URL of the online help page. Please refer to "Online Help Management" for details.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

65

DBCOMBO

■ Example	272
■ Properties	275

The DBCOMBO control represents a filter criterion of a database query. It provides for a value help that is read from the database and a convenient way to append the filter criterion to the SELECT statement. In contrast to DBFIELD, the valid values are read from the database when the page is loaded.

Example

The following image shows an example in which a DBCOMBO control is used for the filter criterion "Country" within a simple business partner report. The combo box options are read from table "COUNTRY" when the page is loaded.

Title	Last Name	First Name	Cou	State
Mr	Strauch	Felix	DE	
Mr	Max	Martin	DE	
Mr	Schumacher	Michael	DE	
MSS	Hahn	Ulla	DE	
Dr	Ahlenfeld	Susanne	DE	Nordrhein-Westfalen

Have a look at the XML layout definition:

```
<rowarea name="Filter Criteria">
  <itr>
    <label name="Country" width="80">
    </label>
    <dbcombo valueprop="dBComboCountry" querycolumn="COUNTRY"
datasource="addressdb"
              valuehelptable="COUNTRY" valuehelpcolumn="ID" width="80">
    </dbcombo>
    <hdist width="100%">
    </hdist>
    <button name="Execute" method="onExecute">
    </button>
  </itr>
</rowarea>
<vdist>
</vdist>
<rowarea name="Result" height="140">
  <itr height="100%">
    <textgridsss2 griddataprop="result" rowcount="5" width="100%">
      <column name="Title" property="TITLE" width="50">
```

```

        </column>
        <column name="Last Name" property="LASTNAME" width="100">
        </column>
        <column name="First Name" property="FIRSTNAME" width="100">
        </column>
        <column name="Country" property="COUNTRY" width="50">
        </column>
        <column name="State" property="STATE" width="100%">
        </column>
    </textgridsss2>
</itr>
</rowarea>

```

The corresponding adapter code is:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.DBCOMBOInfo;
import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBUtil;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBCondition;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;

// This class is a generated one.

public class DBCOMBO_Adapter
    extends Adapter
    implements IDBQUERYConnectionProvider, IDBDemoAdapter
{
    // -----
    // members
    // -----

    private int m_port = 9001;
    private int m_dbPort = 9001;
    private Connection m_connection;
    ...
    DBQUERYStartDatabaseThread m_startDemoDBThread;
    // -----
    // property access
    // -----

    // property >port<
    public void setPort(int value) { m_port = value;}
    public int getPort(){ return m_port;}
}

```

```

// property >dbComboCountry<
DBCComboInfo m_dBComboCountry = new DBCComboInfo(this);
public DBCComboInfo getDBComboCountry() { return m_dBComboCountry; }

// property >result<
DBTEXTGRIDCollection m_result = new DBTEXTGRIDCollection();
public DBTEXTGRIDCollection getResult() { return m_result; }
public void setResult(DBTEXTGRIDCollection value) { m_result = value; }
...
// -----
// public adapter methods
// -----

public Connection getDBConnection(String datasource)
{
    try
    {
        if (m_connection == null)
        {
            String jdbcDriverClassName = "org.hsqldb.jdbcDriver";
            String jdbcUrl = "jdbc:hsqldb:hsqldb://localhost:"+m_dbPort;
            String user = "sa";
            String password = "";
            Class.forName(jdbcDriverClassName);
            m_connection = DriverManager.getConnection(jdbcUrl, user, password);
        }
        return m_connection;
    }
    catch (Exception exc)
    {
        throw new DelegateError(exc);
    }
}
...
public void onExecute()
{
    try
    {
        StringBuffer sb = new StringBuffer();
        sb.append("SELECT * FROM BUSINESSPARTNER INNER JOIN ADDRESS ON ↵
BUSINESSPARTNER.ID =
                                ADDRESS.BUSINESSPARTNERID");
        DBUtil.addToQuery(sb, new IDBCCondition[] { m_dBComboCountry }, true);
        String dataSource = m_dBComboCountry.getDataSource();
        Connection con = getDBConnection(dataSource);
        ResultSet rs = con.createStatement().executeQuery(sb.toString());
        m_result.initWithResultSet(rs);
    }
    catch (Exception exc)
    {
        throw new DelegateError(exc);
    }
}

```

```

    }
}
}

```

The property `dbComboCountry` is of type `DBCOMBOInfo` (from the package `com.softwareag.cis.server.util`). The `DBCOMBOInfo` implements (like all DB controls) the interface `IDBCondition` (`com.softwareag.cis.server.util`). With class `DBUtil` (`com.softwareag.cis.server.util`), you can append values of the filter criteria to the `SELECT` statement in a convenient way. See the JavaDoc documentation of class `DBUtil` for details.

The `DBCOMBOInfo` class does not open a database connection on its own (same to all DB controls). The embedding adapter provides for an implementation of interface `IDBQUERYConnectionProvider` (`com.softwareag.cis.server.util`) when creating a `DBCOMBOInfo` object. The interface method `getDBConnection` is called once - at the first time the `DBCOMBOInfo` has to access the database. There are no updates (insert/update/delete) done with this connection. As the `DBCOMBOInfo` does not open the connection, it does not care about closing the connection.

For displaying the result, the class `DBTEXTGRIDCollection` (from the package `com.softwareag.cis.server.util`) is used. This class extends `TEXTGRIDCollection` by the ability to initialise the collection with a result set (`java.sql.ResultSet`). For each line of the result set, it creates an object of class `DBTEXTGRIDLine` (package `com.softwareag.cis.server.util`). Class `DBTEXTGRIDLine` implements the interface `IDynamicAccess`. With this, "normal" text grid controls can be used to visualize the data of the `DBTEXTGRIDCollection`.

Properties

Basic			
valueprop	Property that returns a <code>DBCOMBOInfo</code> -instance. This instance provides for the combo box options that are read from database as well as for a convenient way to append the filter value to query string.	Obligatory	
querycolumn	Name of the column in the query to that the filter criteria is belongs to. This column may differ from the value help table/column (properties <code>VALUEHELPTABLE</code> , <code>VALUEHELPCOLUMN</code>). This name is used to build a SQL string in method <code>"toSQLString"</code> .	Obligatory	
datasource	Logical identifier of the data source to use. This name is passed to the connection provider in method <code>"IDBQUERYConnectionProvider.getDBConnection"</code> .	Obligatory	
valuehelptable	Name of the table from there the list of valid values can be read.	Obligatory	
valuehelpcolumn	Name of the column from there the list of valid values can be read.	Obligatory	

valuehelpcolumnndescr	Name of a column where an additional description of the valid values is stored. The name must identify a column inside the "value help table" (property VALUEHELPTABLE).	Optional	
size	Integer value that defines the number of lines being displayed. If the SIZE is set to "1", the selection is displayed as combo box. If it is set to ">1", it is displayed as multi line selection.	Optional	
flush	Flushing behaviour\; please view "Common Rules" for details	Optional	screen server
displayonly	If set to "true", the DBCOMBO will not be accessible for input. It is just used as an output field.	Optional	
width	Width of DBCOMBO in pixels or as percentage value.	Optional	100 120 140 160 180 200 50% 100%
align	Explicit Alignment	Optional	left center right
valign	Explicit Alignment	Optional	top middle bottom
colspan	Number of columns occupied by this control.	Optional	1 2 3 4 5 50

			int-value
rowspan	Number of rows occupied by this control.	Optional	1 2 3 4 5 50 int-value
combostyle	Explicit style information passed to the DBCOMBO. Example: if you want the text inside the DBCOMBO to be right aligned, define "text-align: right".	Optional	
helpid	Identifier that is used for building the URL of the online help page. Please refer to "Online Help Management" for details.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

66 DBSELECTOPTION

■ Example	280
■ Properties	284

The DBSELECTOPTION control manages a single filter criterion of a database query. In contrast to the DBFIELD control, it allows the input of several values in a convenient way. Each value is assigned to an operator (e.g. "equals", "like", "between", etc.). The operator-value pairs are linked with a logical "OR" (represented by the string "||"). You can either directly input a string of operator-value pairs into the DBSELECTOPTION, or you invoke the value help. The resulting pop-up displays a grid of operator-value pairs. Maybe this still sounds rather difficult - wait for the following example.

The grid pop-up itself offers a value help for entering appropriate values. The list of valid values (with an optional description) is read from database. The value help table/column (that contains the valid values) can differ from the table on which the query is executed.

Example

The following image shows an example in which the DBSELECTOPTION manages the filter criterion "Title" within a simple business partner report. The report shows partners that have a title equal to "Dr" or equal to "Mr".

The screenshot shows a 'Filter Criteria' dialog box with a dropdown menu set to '=Dr||=Mr' and an 'Execute' button. Below it is a 'Result' table with three columns: Title, Last Name, and First Name. The table contains five rows of data.

Title	Last Name	First Name
Mr	Strauch	Felix
Mr	Max	Martin
Mr	Schumacher	Michael
Mr	Schumacher	Ralf
Dr	Ahlenfeld	Susanne

On value help request, the following pop-up appears:

Operator	Value
=	Dr
=	Mr

OK

The first column defines the operator, the second the filter value. The value help of the first column returns a list of valid operators. The second column has value help, too (see properties `valuehelptable` and `valuehelpcolumn`). The following pop-up shows the list of valid values read from the table "TITLE".

Operator	Value
=	Dr
=	Dr
	Mr
	Mss

OK

Have a look at the XML layout definition:

```
<rowarea name="Filter Criteria">
  <itr>
    <label name="Title" width="60">
    </label>
    <dbselectoption querycolumn="TITLE" datasource="addressdb"
                    valueprop="dbSelectOptionTitle" valuehelptable="TITLE"
                    valuehelpcolumn="ID">
    </dbselectoption>
    <hdist width="100%">
    </hdist>
    <button name="Execute" method="onExecute">
    </button>
  </itr>
</rowarea>
<vdist>
</vdist>
<rowarea name="Result" height="140">
  <itr height="100%">
    <textgridsss2 griddataprop="result" rowcount="5" width="100%">
      <column name="Title" property="TITLE" width="20%">
      </column>
      <column name="Last Name" property="LASTNAME" width="40%">
      </column>
      <column name="First Name" property="FIRSTNAME" width="40%">
      </column>
    </textgridsss2>
  </itr>
</rowarea>
```

The corresponding adapter code is:

```
package com.softwareag.cis.test20;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.DBSELECTOPTIONInfo;
import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBUtil;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBCondition;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;

public class DBSELECTOPTIONDemoAdapter
  extends Adapter
  implements IDBQUERYConnectionProvider
{
  // -----
  // members
  // -----
}
```

```

Connection m_connection;

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

// property >dbselectOptionTitle<
DBSELECTOPTIONInfo m_dBSelectOptionTitle = new DBSELECTOPTIONInfo(this, this);
public DBSELECTOPTIONInfo getDBSelectOptionTitle() { return ↵
m_dBSelectOptionTitle; }

// property >result<
DBTEXTGRIDCollection m_result = new DBTEXTGRIDCollection();
public DBTEXTGRIDCollection getResult() { return m_result; }
public void setResult(DBTEXTGRIDCollection value) { m_result = value; }

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

/** implementation of interface IDBQUERYConnectionProvider */
public Connection getDBConnection(String datasource)
{
    if (m_connection == null)
    {
        try
        {
            Class.forName("org.hsqldb.jdbcDriver");
            m_connection = DriverManager.getConnection("jdbc:hsqldb:hsqldb://localhost", ↵
"sa", "");
        }
        catch (Exception exc)
        {
            throw new DelegateError(exc);
        }
    }
    return m_connection;
}

/** executes the query with current values of the filter criteria "Title". */
public void onExecute()
{
    try
    {
        StringBuffer sb = new StringBuffer();
        sb.append("SELECT * FROM BUSINESSPARTNER");
        DBUtil.addToQuery(sb, new IDBCondition[] { m_dBSelectOptionTitle}, true);

        String dataSource = m_dBSelectOptionTitle.getDataSource();
        Connection con = getDBConnection(dataSource);

```

```
ResultSet rs = con.createStatement().executeQuery(sb.toString());
    m_result.initWithResultSet(rs);
}
catch (Exception exc)
{
    throw new DelegateError(exc);
}
}
```

The adapter property `dbselectOptionTitle` is of type `DBSELECTOPTIONInfo` (from the package `com.softwareag.cis.server.util`). The `DBSELECTOPTIONInfo` implements (like all DB controls) the interface `IDBCondition` (`com.softwareag.cis.server.util`). On report execution, you may use the class `DBUtil` (`com.softwareag.cis.server.util`) to append the value of property `dbselectOptionTitle` to the SQL query. See the JavaDoc documentation of class `DBSELECTOPTIONInfo` and `DBUtil` for details.

The `DBSELECTOPTIONInfo` class does not open a database connection on its own (same to all DB controls). The embedding adapter provides for an implementation of interface `IDBQUERYConnectionProvider` (`com.softwareag.cis.server.util`) when creating a `DBSELECTOPTIONInfo` object. The interface method `getDBConnection` is called once - at the first time the `DBSELECTOPTIONInfo` has to access the database.

There are no updates (insert/update/delete) done with this connection. As the `DBSELECTOPTIONInfo` does not open the connection, it does not care about closing the connection.

For displaying the result, the class `DBTEXTGRIDCollection` (from the package `com.softwareag.cis.server.util`) is used. This class extends `TEXTGRIDCollection` by the ability to initialise the collection with a result set (`java.sql.ResultSet`). For each line of the result set, it creates an object of class `DBTEXTGRIDLine` (package `com.softwareag.cis.server.util`). Class `DBTEXTGRIDLine` implements the interface `IDynamicAccess`. With this, "normal" text grid controls can be used to visualize the data of the `DBTEXTGRIDCollection`.

Properties

Basic			
querycolumn	Name of the column in the query to that the filter criteria belongs to. This column may differ from the value help table/column (properties <code>VALUEHELPTABLE</code> , <code>VALUEHELPCOLUMN</code>). This name is used to build a SQL string in method <code>"toSQLString"</code> .	Obligatory	
datasource	Logical identifier of the data source to use. This name is passed to the connection provider in method <code>"IDBQUERYConnectionProvider.getDBConnection"</code> .	Obligatory	

valueprop	Property that returns a DBSELECTOPTIONInfo -instance. This instance provides for the value help read from database as well as for a convenient way to append the filter value to query string.	Obligatory	
valuehelptable	Name of the table from where the valid values of the DBSELECTOPTION control are stored.	Optional	
valuehelpcolumn	Name of the column from where the valid values of the DBSELECTOPTION control are stored.	Optional	
valuehelpcolumndescr	Name of a column where an additional description of the valid values is stored. The column must be inside the "value help table" (property VALUEHELPTABLE).	Optional	
width	Width of DBFIELD in pixels or as percentage value.	Optional	
length	Width of DBFIELD in amount of characters. WIDTH and LENGTH should not be used together.	Optional	
datatype	<p>The DBSELECTOPTION control manages multiple "operator-value"-pairs.</p> <p>By default, each is managed as string. By the DATATYPE property, force the type of the data that is represented. As a consequence the DBFIELD controls inside the "operator-value"-popup is checking the data during input (e.g. if the DATATYPE is "int", it is not allowed to enter alphabetic characters) and adds a logic to transfer the data into various output formats (e.g. if the DATATYPE is "date", the date is formatted into the right date format). In addition it displays a standard "value help" popup dialog for some data types (e.g. if the DATATYPE is "date" then automatically a date input dialog pops up if invoking "value help").</p>	Optional	int float date
flush	Flushing behaviour, please view "Common Rules" for details	Optional	screen server
displayonly	If set to "true", the DBFIELD will not be accessible for input. It is just used as an output field.	Optional	true false
align	Explicit Alignment	Optional	
valign	Explicit Alignment	Optional	
colspan	Number of columns occupied by this control.	Optional	
rowspan	Number of rows occupied by this control.	Optional	
fieldstyle	<p>Explicit style information passed to the DBFIELD.</p> <p>Example: if you want the text inside the DBFIELD to be right aligned, define "text-align: right".</p>	Optional	

helpid	Identifier that is used for building the URL of the online help page. Please refer to "Online Help Management" for details.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

67

DBCHECKBOX

▪ Example	288
▪ Properties	290

The DBCHECKBOX control represents a filter criterion of a database query of type Boolean. Compared to a “normal” CHECKBOX control, the only benefit is the convenient way of appending the filter criterion to the SELECT statement.

Example

The following image shows an example in which the DBCHECKBOX is used for the filter criterion “Female”:

The screenshot shows a window titled "Filter Criteria" with a text field containing "Female" and a checked checkbox. To the right is an "Execute" button. Below this is a "Result" section containing a table with the following data:

Title	Last Name	First Name	Cou	State
MSS	Hahn	Ulla	DE	
Dr	Ahlenfeld	Susanne	DE	Nordrhein-Westfalen
Dr	Ahlenfeld	Susanne	DE	Nordrhein-Westfalen

Have a look at the XML layout definition:

```
<rowarea name="Filter Criteria">
  <itr>
    <label name="Female" width="80">
    </label>
    <dbcheckbox valueprop="dbCheckboxFemale" querycolumn="FEMALE">
    </dbcheckbox>
    <hdist width="100%">
    </hdist>
    <button name="Execute" method="onExecute">
    </button>
  </itr>
</rowarea>
<vdist>
</vdist>
<rowarea name="Result" height="140">
  <itr height="100%">
    <textgridsss2 griddataprop="result" rowcount="5" width="100%">
      <column name="Title" property="TITLE" width="50">
      </column>
      <column name="Last Name" property="LASTNAME" width="100">
      </column>
      <column name="First Name" property="FIRSTNAME" width="100">
      </column>
    </textgridsss2>
  </itr>
</rowarea>
```

```

        <column name="Country" property="COUNTRY" width="50">
        </column>
        <column name="State" property="STATE" width="100%">
        </column>
    </textgridsss2>
</itr>
</rowarea>

```

The corresponding adapter code is:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.DBCHECKBOXInfo;
import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBUtil;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBCondition;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;

// This class is a generated one.

public class DBCHECKBOX_Adapter
    extends Adapter
    implements IDBQUERYConnectionProvider, IDBDemoAdapter
{
    // -----
    // members
    // -----
    ...
    // property >dBCheckboxFemale<
    DBCHECKBOXInfo m_dBCheckboxFemale = new DBCHECKBOXInfo();
    public DBCHECKBOXInfo getDBCcheckboxFemale() { return m_dBCheckboxFemale; }

    // property >result<
    DBTEXTGRIDCollection m_result = new DBTEXTGRIDCollection();
    public DBTEXTGRIDCollection getResult() { return m_result; }
    public void setResult(DBTEXTGRIDCollection value) { m_result = value; }
    ...
    // -----
    // public adapter methods
    // -----
    ...
    public void onExecute()
    {
        try
        {
            StringBuffer sb = new StringBuffer();
            sb.append("SELECT * FROM BUSINESSPARTNER INNER JOIN ADDRESS ON ↵
BUSINESSPARTNER.ID =

```

```
ADDRESS.BUSINESSPARTNERID");
    DBUtil.addToQuery(sb, new IDBCondition[] { m_dBCheckboxFemale }, true);
    Class.forName("org.hsqldb.jdbcDriver");
    Connection con = ↵
DriverManager.getConnection("jdbc:hsqldb:hsqldb://localhost", "sa", "");
    ResultSet rs = con.createStatement().executeQuery(sb.toString());
    m_result.initWithResultSet(rs);
    }
    catch (Exception exc)
    {
        throw new DelegateError(exc);
    }
}
```

The adapter property `dBCheckboxFemale` is of type `DBCHECKBOXInfo` (from the package `com.softwareag.cis.server.util`). This class has a property value that is manipulated by the `DBCHECKBOX` control. The `DBCHECKBOXInfo` implements (like all DB controls) the interface `IDBCondition` (`com.softwareag.cis.server.util`). With class `DBUtil` (`com.softwareag.cis.server.util`) there is a convenient way to append the filter value to the SQL query. See the JavaDoc documentation of class `DBCHECKBOXInfo` and `DBUtil` for details.

For displaying the result, the class `DBTEXTGRIDCollection` (from the package `com.softwareag.cis.server.util`) is used. This class extends `TEXTGRIDCollection` by the ability to initialise the collection with a result set (`java.sql.ResultSet`). For each line of the result set, it creates an object of class `DBTEXTGRIDLine` (package `com.softwareag.cis.server.util`). Class `DBTEXTGRIDLine` implements the interface `IDynamicAccess`. With this, “normal” text grid controls can be used to visualize the data of the `DBTEXTGRIDCollection`.

Properties

Basic			
valueprop	Property that returns a <code>DBCHECKBOXInfo</code> -instance. With class <code>DBUtil</code> this instance provides for a convenient way to append the filter value to query string.	Obligatory	
querycolumn	Name of the column in the query to that the filter criteria belongs to. This column may differ from the value help table/column (properties <code>VALUEHELPTABLE</code> , <code>VALUEHELPCOLUMN</code>). The name is used to build a SQL string in method <code>"toSQLString"</code> .	Obligatory	
flush	Flush reaction as described in "Common Rules"	Optional	screen server
displayonly	If set to "true", the displayed checkbox can not be changed. The default is "false".	Optional	true false

width	Width of the control in pixels or as percentage value. This does not change the size of the "box" of the CHECKBOX, but the size of the column in which the CHECKBOX is located.	Optional	100 120 140 160 180 200 50% 100%
align	Explicit Alignment	Optional	left center right
valign	Explicit Alignment	Optional	top middle bottom
colspan	Number of columns occupied by this control.	Optional	1 2 3 4 5 50 int-value
rowspan	Number of rows occupied by this control.	Optional	1 2 3 4 5 50

			int-value
helpid	Identifier that is used for building the URL of the online help page. Please refer to "Online Help Management" for details.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

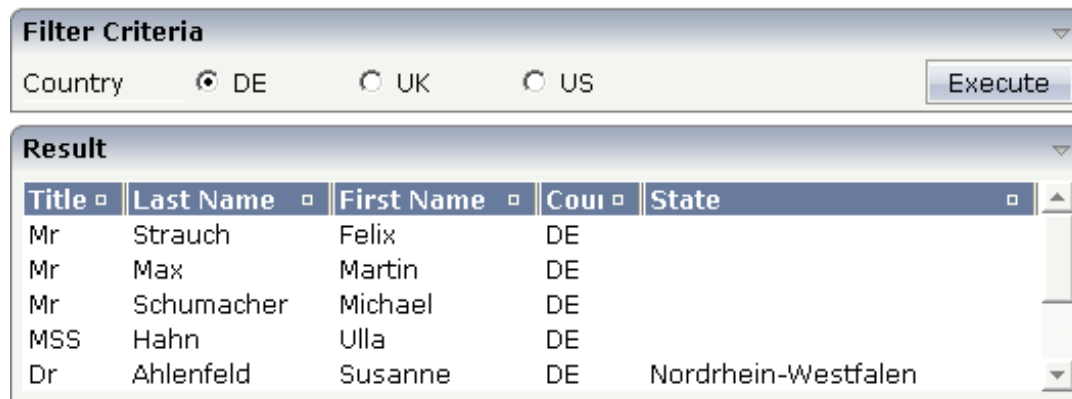
68 DBRADIOBUTTON

■ Example	294
■ Properties	297

The DBRADIOBUTTON control represents a filter criterion of a database query. Compared to a “normal” RADIOBUTTON control, the only benefit is the convenient way of appending the filter criterion to the SELECT statement.

Example

The following image shows an example in which the DBRADIOBUTTON is used for the filter criterion “Country”:



Title	Last Name	First Name	Cou	State
Mr	Strauch	Felix	DE	
Mr	Max	Martin	DE	
Mr	Schumacher	Michael	DE	
MSS	Hahn	Ulla	DE	
Dr	Ahlenfeld	Susanne	DE	Nordrhein-Westfalen

Have a look at the XML layout definition:

```
<rowarea name="Filter Criteria">
  <itr>
    <label name="Country" width="80">
    </label>
    <dbradiobutton valueprop="dBRadioButtonSex" querycolumn="COUNTRY" value="DE">
    </dbradiobutton>
    <hdist>
    </hdist>
    <label name="DE" asplaintext="true">
    </label>
    <hdist width="40">
    </hdist>
    <dbradiobutton valueprop="dBRadioButtonSex" querycolumn="COUNTRY" value="UK">
    </dbradiobutton>
    <hdist>
    </hdist>
    <label name="UK" asplaintext="true">
    </label>
    <hdist width="40">
    </hdist>
    <dbradiobutton valueprop="dBRadioButtonSex" querycolumn="COUNTRY" value="US">
    </dbradiobutton>
    <hdist>
  </itr>
</rowarea>
```



```

        </hdist>
        <label name="US" asplaintext="true">
        </label>
        <hdist>
        </hdist>
        <hdist width="100%">
        </hdist>
        <button name="Execute" method="onExecute">
        </button>
    </itr>
</rowarea>
<vdist>
</vdist>
<rowarea name="Result" height="140">
    <itr height="100%">
        <textgridsss2 griddataprop="result" rowcount="5" width="100%">
            <column name="Title" property="TITLE" width="50">
            </column>
            <column name="Last Name" property="LASTNAME" width="100">
            </column>
            <column name="First Name" property="FIRSTNAME" width="100">
            </column>
            <column name="Country" property="COUNTRY" width="50">
            </column>
            <column name="State" property="STATE" width="100%">
            </column>
        </textgridsss2>
    </itr>
</rowarea>

```

The corresponding adapter code is:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.DBRADIOBUTTONInfo;
import com.softwareag.cis.server.util.DBTEXTGRIDCollection;
import com.softwareag.cis.server.util.DBUtil;
import com.softwareag.cis.server.util.DelegateError;
import com.softwareag.cis.server.util.IDBCondition;
import com.softwareag.cis.server.util.IDBQUERYConnectionProvider;

// This class is a generated one.

public class DBRadioButton_Adapter
    extends Adapter
    implements IDBQUERYConnectionProvider, IDBDemoAdapter
{
    ...
    // property >dbRadioButtonSex<

```

```
DBRADIOBUTTONInfo m_dBRadioButtonSex = new DBRADIOBUTTONInfo();
public DBRADIOBUTTONInfo getDBRadioButtonSex() { return m_dBRadioButtonSex; }

// property >result<
DBTEXTGRIDCollection m_result = new DBTEXTGRIDCollection();
public DBTEXTGRIDCollection getResult() { return m_result; }
public void setResult(DBTEXTGRIDCollection value) { m_result = value; }
...
public void onExecute()
{
    try
    {
        StringBuffer sb = new StringBuffer();
        sb.append("SELECT * FROM BUSINESSPARTNER INNER JOIN ADDRESS ON ↵
BUSINESSPARTNER.ID =
                                ADDRESS.BUSINESSPARTNERID");
        DBUtil.addToQuery(sb, new IDBCondition[] { m_dBRadioButtonSex }, true);
        Class.forName("org.hsquidb.jdbcDriver");
        Connection con = ↵
DriverManager.getConnection("jdbc:hsquidb:hsquidb://localhost", "sa", "");
        ResultSet rs = con.createStatement().executeQuery(sb.toString());
        m_result.initWithResultSet(rs);
    }
    catch (Exception exc)
    {
        throw new DelegateError(exc);
    }
}
...
}
```

The adapter property `dBRadioButtonCountry` is of type `DBRADIOBUTTONInfo` (from the package `com.softwareag.cis.server.util`). This class has a property value that is manipulated by the `DBRADIOBUTTON` control. The `DBRADIOBUTTONInfo` implements (like all DB controls) the interface `IDBCondition` (`com.softwareag.cis.server.util`). With class `DBUtil` (`com.softwareag.cis.server.util`), there is a convenient way to append the filter value to the SQL query. See the JavaDoc documentation of class `DBRADIOBUTTONInfo` and `DBUtil` for details.

For displaying the result, the class `DBTEXTGRIDCollection` (from the package `com.softwareag.cis.server.util`) is used. This class extends `TEXTGRIDCollection` by the ability to initialise the collection with a result set (`java.sql.ResultSet`). For each line of the result set, it creates an object of class `DBTEXTGRIDLine` (package `com.softwareag.cis.server.util`). Class `DBTEXTGRIDLine` implements the interface `IDynamicAccess`. With this, “normal” text grid controls can be used to visualize the data of the `DBTEXTGRIDCollection`.

Properties

Basic			
valueprop	Property that returns a DBRADIOBUTTONInfo -instance. With class DBUtil this instance provides for a convenient way to append the filter value to query string.	Obligatory	
querycolumn	Name of the column in the query to that the filter criteria belongs to. This column may differ from the value help table/column (properties VALUEHELPTABLE, VALUEHELPCOLUMN). The name is used to build a SQL string in method "toSQLString".	Obligatory	
value	Value that represents this instance of the DBRADIOBUTTON control. The value is set into the adapter property that is defined by the VALUEPROP attribute when the user clicks onto the control. - Vice versa: the control is switched to "marked" when the adapter property holds the value defined.	Obligatory	
displayonly	If set to "true", the displayed checkbox can not be changed. The default is "false".	Optional	true false
flush	Flush reaction as described in "Common Rules"	Optional	screen server
width	Width of the control in pixels or as percentage value. This does not change the size of the "box" of the CHECKBOX, but the size of the column in which the CHECKBOX is located.	Optional	100 120 140 160 180 200 50% 100%
align	Explicit Alignment	Optional	left center right
valign	Explicit Alignment	Optional	top

			middle bottom
colspan	Number of columns occupied by this control.	Optional	1 2 3 4 5 50 int-value
rowspan	Number of rows occupied by this control.	Optional	1 2 3 4 5 50 int-value
helpid	Identifier that is used for building the URL of the online help page. Please refer to "Online Help Management" for details.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

XI

Personalization of Pages

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

Goal

Customized Layout - Concepts

Customized Layout - Example

Customized Proposals - Concepts

Customized Proposals - Example

69

Goal

Developers of standard enterprise applications are confronted with the problem that pages of their application need to be customized to a certain degree by their customers. Customizing includes two aspects:

- **Customized Layout**

A standard page typically offers more features and fields than are required by the specific customer. Consequently, application developers have to make their pages flexible in order to allow the switching off/on of certain information and functions on the screen.

- **Customized Proposals**

Certain input parameters of your application can be proposed automatically to the user. If, for example, your application deals with currencies and you have a default currency to be used, this default currency can be proposed as input automatically without having the user to provide input all the time.

This is what we call the “personalization” of pages.

Depending on the size and structure of a specific customer application, the personalization may be done several times. Maybe one page looks different in each business unit of the customer, and maybe one page even looks different within several departments of one business unit.

The goal of the personalization framework is to keep - as much as possible - work away from the application development being responsible for the layout of the page and for the development of corresponding adapter objects. The focus of adapter objects clearly is to pass information from the user input into the business logic behind the adapters, and vice versa.

The framework described in this part is an offer - it bases on a generalized filter model for information passed between adapter and page. The framework is an optional offer to be used by you - you also can build up own frameworks or do not use personalization at all. If not used, there are no negative impacts in means of performance and resource consumption.

The framework is released in several steps. The focus of the first step (which is described here) is to control the visibility of controls on pages and to automate the proposal of input values. This part first introduces the concepts for a **customized layout** (with an **example**), and then the concepts for **customized proposals** (also with an **example**). Both concepts can be used independently from one another.

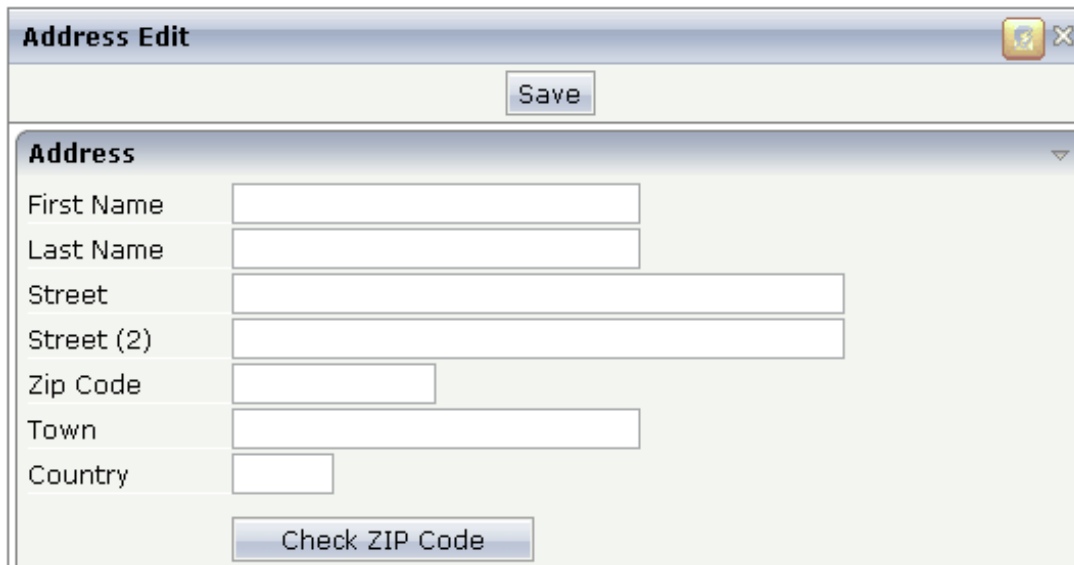
70

Customized Layout - Concepts

■ Overview	304
■ Dynamic Controls	306
■ Using Filters	306
■ Personalization Filter	307
■ Personalization Scenario Sequence	308
■ Maintaining Personalization Data	308
■ Persisting Personalization Data	308

Overview

Let us approach the framework from the result side. The following screen is a simple address maintenance:



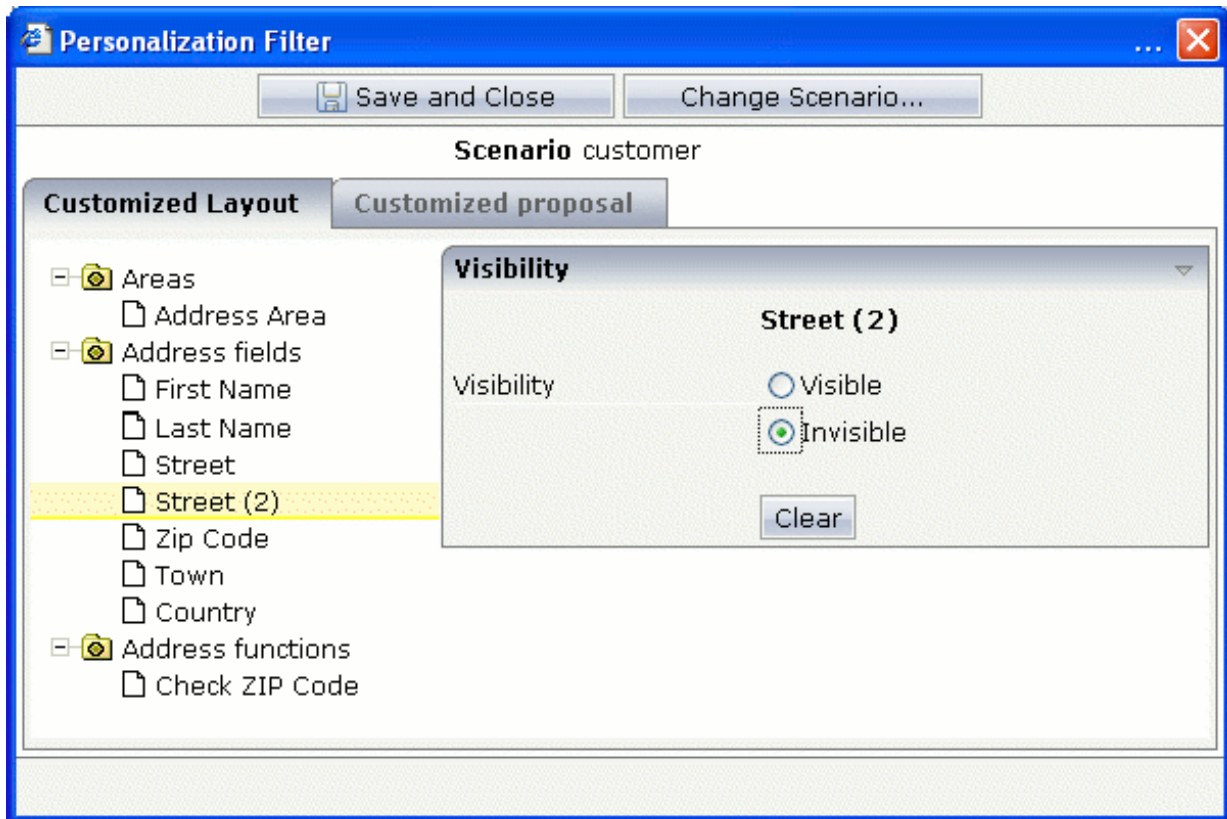
The screenshot shows a window titled "Address Edit" with a close button (X) in the top right corner. Below the title bar is a "Save" button. The main content area is titled "Address" and contains the following fields:

- First Name
- Last Name
- Street
- Street (2)
- Zip Code
- Town
- Country

At the bottom of the form is a "Check ZIP Code" button.

You see that in the right top corner there is a special icon directly to left of the close icon. This “personalization icon” is not always visible. It is only visible, if a certain parameter is defined inside the session context (e.g. when an administrator logs on).

When choosing the icon, a window appears:



The window shows on the left a tree of personalizable aspects. When selecting a node, the aspect can be edited on the right. In our example, we set the "Street (2)" aspect and the "Check ZIP Code" aspect to be invisible. Make sure that the personalization is done within a certain scenario (in this example, the scenario has the name "customer").

After choosing the **Save and Close** button, the address page now looks as follows:

You see that two lines have disappeared. This definition and the corresponding screen layout are now automatically valid for all users that work in the personalization scenario "customer". Maybe other users work in a different scenario and thus see a different screen layout.

The framework basics will be explained below, and the implementation of the above address example will be shown in detail.

Dynamic Controls

The basis of all personalization is the possibility to define a control's rendering and behavior dynamically via the adapter. Each `BUTTON` control, for example, provides for a `visibleprop` property by which the adapter can specify whether the button is shown or not.

The control of visibility was much extended for personalization purposes. It is possible to control visibility for all major controls including:

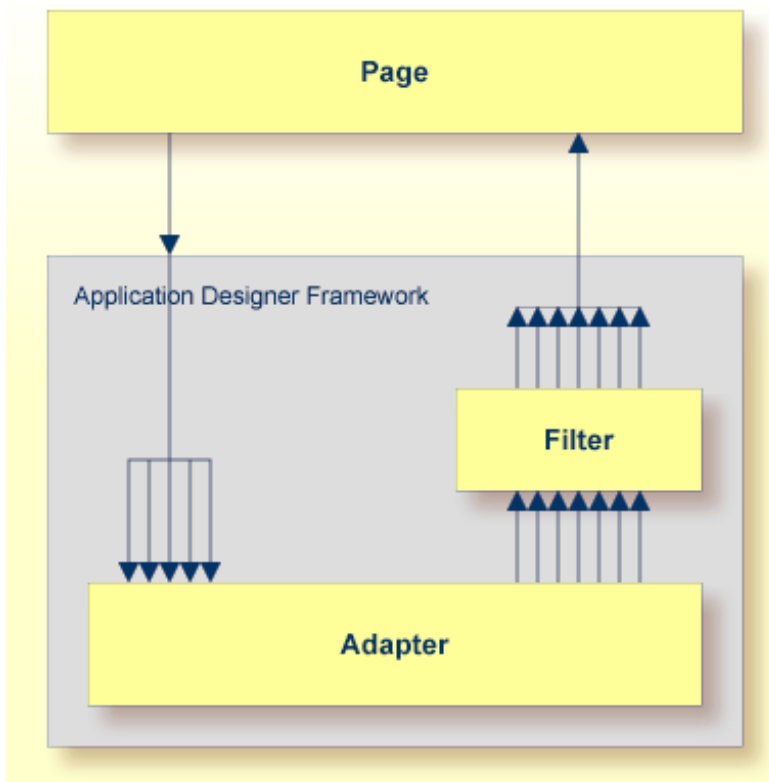
- TR and ITR rows.
- ROWAREA, ROWTABLE0, and other container controls.
- BUTTON, ICON, MENUBUTTON, FIELD, and other input controls

By defining the `visibleprop` properties in the controls, you can - without using the next framework parts - already control the page layout dynamically in a detailed way. It is up to you to specify where and how to use the `visibleprop` definitions. Sometimes it is useful to use the definitions on single elements (e.g. `BUTTON` controls). Sometimes it is useful to define the visibility on row level (ITR, TR control). If inside your screen there is the combination of "label and field in one row", then it makes sense to define visibility on row level.

Using Filters

Having defined the `visibleprop` definitions inside the page, the question now is how to provide for the corresponding property values at runtime.

The concept of filters offers a flexible and efficient way to provide for the values in front of the adapter logic - without even letting the adapter logic know about.



After the Application Designer framework has collected the property values from the adapter, it passes them to a filter that may be assigned to an adapter. The filter can now manipulate the values. This means it can

- change the value of properties,
- add properties that are not provided for by the adapter.

What does this mean for personalization? It means that the adapter does not need to implement all the `visibleprop` properties on its own but can “outsource” this task to a generic filter.

Personalization Filter

The personalization filter is a concrete filter implementation that is responsible for providing personalization data values. On the one hand, the personalization filter is kept well apart from the Application Designer framework; i.e. you will not find any specific personalization aspects inside the base framework of Application Designer (e.g. there are no personalization methods inside the `Adapter` class). On the other hand, it is tightly integrated into the Application Designer framework - being a concrete implementation of open interfaces that are offered.

The main aspects of the personalization filter are:

- Inside the page layout definition, a developer decides at design time which properties are controlled by the filter. These properties are called “managed properties” in the following text.
- At runtime, the filter provides for the property values depending on so-called personalization scenarios: values for the managed properties are kept with reference to a scenario in a persistent storage. This means that the property values are either stored in an SQL database or in the file system.
- At runtime (e.g. when a user logs on to an application system), the application defines the active personalization scenario(s) in the session context. Depending on the scenario(s), the right filter information is read from persistent storage and used by the filter.

Personalization Scenario Sequence

It is possible to define a sequence of personalization scenarios at runtime. Maybe a user is assigned after having logged on to the scenarios "Department 12", "Business Unit1". In this case, the personalization filter first tries to find filter information for "Department 12"; if filter information is not defined, then it tries to find filter information for "Business Unit1".

Consequently, you can set up a flexible way to try to provide for the most detailed personalization definition for specific screens - and referencing to a general personalization definition (or none at all) for other screens.

Maintaining Personalization Data

Personalization data is maintained through a specific page. The maintenance page may be accessed directly from a personalized page via a certain control - a certain icon in the title bar. This icon is only visible when a certain context parameter is set. It should not be visible during normal user sessions, but should only be available for administrative sessions.

Persisting Personalization Data

The personalization data is stored in the local file system, inside the directory */pers* of an Application Designer project.



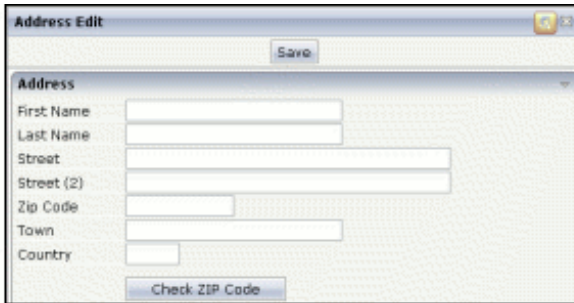
Caution: Filter information is runtime information, not design time information. This means: filter information is not part, for example, of a WAR file that you deliver, it is maintained within a delivered system by your customer.

71

Customized Layout - Example

■ XML Layout	310
■ Java Adapter Code	312

Let us take the [address maintenance example](#) that was shown in the previous section.

A screenshot of a web application window titled "Address Edit". The window has a title bar with a "Save" button. Below the title bar is a section labeled "Address" with a dropdown arrow. Inside this section are several text input fields: "First Name", "Last Name", "Street", "Street (2)", "Zip Code", "Town", and "Country". Below these fields is a "Check ZIP Code" button. The background of the form area has a light gray grid pattern.

XML Layout

The XML layout looks as follows:

```
<page model="CustomizedLayoutAdapter">
  <titlebar name="Address Edit">
    <persedit persprop="paInfo">
      </persedit>
    </titlebar>
    <header withdistance="false">
      <button name="Save" method="onSave">
        </button>
      </header>
    <pagebody>
      <rowarea name="Address" visibleprop="adressAreaVisible">
        <itr visibleprop="firstNameVisible">
          <label name="First Name" width="100">
            </label>
          <field valueprop="firstName" width="200">
            </field>
          </itr>
        <itr visibleprop="lastNameVisible">
          <label name="Last Name" width="100">
            </label>
          <field valueprop="lastName" width="200">
            </field>
          </itr>
        <itr visibleprop="streetVisible">
          <label name="Street" width="100">
            </label>
          <field valueprop="street" width="300">
            </field>
          </itr>
        <itr visibleprop="street2Visible">
          <label name="Street (2)" width="100">
            </label>
          </itr>
        </rowarea>
      </pagebody>
    </header>
  </titlebar>
</page>
```



```

        <field valueprop="street2" width="300">
        </field>
    </itr>
    <itr visibleprop="zipCodeVisible">
        <label name="Zip Code" width="100">
        </label>
        <field valueprop="zipCode" width="100">
        </field>
    </itr>
    <itr visibleprop="townVisible">
        <label name="Town" width="100">
        </label>
        <field valueprop="town" width="200">
        </field>
    </itr>
    <itr visibleprop="countryVisible">
        <label name="Country" width="100">
        </label>
        <field valueprop="country" width="50">
        </field>
    </itr>
    <vdist height="10">
    </vdist>
    <itr visibleprop="checkZipCodeVisible">
        <hdist width="100">
        </hdist>
        <button name="Check ZIP Code" method="onCheckZIPCode">
        </button>
    </itr>
    </rowarea>
</pagebody>
<statusbar withdistance="false">
</statusbar>
<personalization>
    <persproposal property="town" comment="Town">
    </persproposal>
    <persproposal property="country" comment="Country">
    </persproposal>
    <persfilter property="adressAreaVisible" group="Areas" comment="Address Area">
    </persfilter>
    <persfilter property="firstNameVisible" group="Address fields" comment="First ↵
Name">
    </persfilter>
    <persfilter property="lastNameVisible" group="Address fields" comment="Last ↵
Name">
    </persfilter>
    <persfilter property="streetVisible" group="Address fields" comment="Street">
    </persfilter>
    <persfilter property="street2Visible" group="Address fields" comment="Street ↵
(2)">
    </persfilter>
    <persfilter property="zipCodeVisible" group="Address fields" comment="Zip ↵

```

```
Code">
    </persfilter>
    <persfilter property="townVisible" group="Address fields" comment="Town">
    </persfilter>
    <persfilter property="countryVisible" group="Address fields" ↵
comment="Country">
    </persfilter>
    <persfilter property="checkZipCodeVisible" group="Address functions"
        comment="Check ZIP Code">
    </persfilter>
</personalization>
</page>
```

What are the personalization aspects of the XML layout?

- In the TITLEBAR definition, you see a special PERSEDIT control. This control is rendered as an icon. It is only available if the session context indicates an administrative session (details will be provided later). When choosing this icon, the personalization maintenance appears. The control references to a PERSPROP property `paInfo`.
- In the various ITR definitions (each ITR holding a label and a field), there are references to VISIBLEPROP properties.
- In the PERSONALIZATION section, there is a list of PERSFILTER definitions. Each definition indicates a personalizable property and holds some additional information: a comment and a group. The group is used to structure the properties in a tree inside the personalization maintenance page. The comment is used as text for the property. Since personalization is not an end-user task but an administrative task, group and comment are not language-dependent and should be kept in the default language of your application.

Java Adapter Code

The adapter code is:

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

import java.util.Iterator;
import java.util.Properties;

import com.softwareag.cis.pers.Personalization;
import com.softwareag.cis.pers.PersonalizationAdapterInfo;
import com.softwareag.cis.pers.PersonalizationScenarioSequence;
import com.softwareag.cis.server.Adapter;

public class CustomizedLayoutAdapter
    extends Adapter
{
    PersonalizationAdapterInfo m_paInfo = new PersonalizationAdapterInfo(this);
```

```

String m_lastName;
String m_firstName;
String m_street;
String m_street2;
String m_zipCode;
String m_town;
String m_country;

public PersonalizationAdapterInfo getPaInfo() { return m_paInfo; }

public String getLastName() { return m_lastName; }
public void setLastName(String value) { m_lastName = value; }

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

public String getStreet() { return m_street; }
public void setStreet(String value) { m_street = value; }

public String getStreet2() { return m_street2; }
public void setStreet2(String value) { m_street2 = value; }

public String getZipCode() { return m_zipCode; }
public void setZipCode(String value) { m_zipCode = value; }

public String getTown() { return m_town; }
public void setTown(String value) { m_town = value; }

public String getCountry() { return m_country; }
public void setCountry(String value) { m_country = value; }
...
/** initialisation - called when creating this instance*/
public void init()
{
    // switch maintenance on
    Personalization.switchPersonalizationMaintenanceOn(findSessionContext());
    // set up scenarios
    PersonalizationScenarioSequence pss = new ↵
PersonalizationScenarioSequence("customer");
    Personalization.defineScenarioSequenceInContext(findSessionContext(),pss);
    // transfer proposal values
    m_paInfo.applyProposals(this);
}
...
}

```

You see that personalization does not affect the adapter too much:

- There is a member (`m_paInfo`) of type `PersonalizationAdapterInfo` that is made accessible via `getPAInfo()`. This property passes certain information about personalization to the PERSEDIT control inside the page.

- In the `init()` method, there is the call of the method `Personalization.switchPersonalization-MaintenanceOn()`. As parameter, the session context is passed. This method specifies in a certain session context parameter that the current session is a session in which you want to maintain personalization data. It should normally be called during a certain logon page of your application in which you decide by certain logon parameters that now the administrative user logs on in a special administrative mode.
- In the `init()` method, there is the setting of the current personalization scenario. In the example, the scenario sequence contains one scenario: the "customer" scenario. Scenarios are just names that are used as references into the personalization at runtime. The setting of the scenario sequence normally should happen also as part of the logon procedure to your application.

All the rest (the filtering of properties, the calling of the maintenance pop-up, the storing of personalization data) is done automatically. You, the developer, do not have to take care of it.

72

Customized Proposals - Concepts

■ Overview	316
■ Properties Used for Proposals	318
■ Personalization Scenario, Personalization Scenario Sequence	318

Overview

The previous [address maintenance example](#) also provides the option to maintain proposal values. When choosing the personalization icon, a window appears. The second tab area of this window looks as follows:

The screenshot shows a window titled "Personalization Filter" with a blue title bar. Inside, there are two buttons: "Save and Close" and "Change Scenario...". Below these is a label "Scenario customer". There are two tabs: "Customized Layout" and "Customized proposal", with the latter being selected. The "Customized proposal" tab contains a table with two columns: "Name" and "Value". The table has 10 rows. The first two rows are pre-filled with "Country" and "Town". The remaining eight rows are empty. The table has a vertical scrollbar on the right side.

Name	Value
Country	
Town	

Let us assume that the user typically enters addresses of inhabitants of the German city Berlin - maybe the user works in an office for public administration. In this case, the user prefers that the screen proposes the following values:

The screenshot shows a window titled "Personalization Filter" with a blue title bar. Inside, there are two buttons at the top: "Save and Close" and "Change Scenario...". Below these, the text "Scenario customer" is displayed. There are two tabs: "Customized Layout" and "Customized proposal", with the latter being selected. The "Customized proposal" tab contains a table with two columns: "Name" and "Value". The table has several rows, with the first two containing data: "Country" with value "D" and "Town" with value "Berlin". The rest of the rows are empty.

Name	Value
Country	D
Town	Berlin

The next time the address maintenance will be started, it will automatically provide the adequate values:

The screenshot shows a window titled "Address Edit" with a light blue title bar. Inside, there is a "Save" button at the top. Below it, there is a section titled "Address" with a dropdown arrow. This section contains several input fields: "First Name", "Last Name", "Street", "Street (2)", "Zip Code", "Town", and "Country". The "Town" field is pre-filled with "Berlin" and the "Country" field is pre-filled with "D". At the bottom of the "Address" section, there is a "Check ZIP Code" button.

Address	
First Name	<input type="text"/>
Last Name	<input type="text"/>
Street	<input type="text"/>
Street (2)	<input type="text"/>
Zip Code	<input type="text"/>
Town	Berlin
Country	D

Check ZIP Code

Properties Used for Proposals

The concept behind proposals is fairly simple:

- Proposal values are kept for a certain personalization scenario.
- Proposal values can be taken over into a certain adapter in two ways:
 - **Automated Transfer**
Call an `apply()` method when initialising an adapter. In the `apply()` method, the proposed values are automatically transferred into the properties of an adapter by serving the set property methods or going through the `IDynamicAccess` interface.
 - **Application Transfer**
The application can ask for the proposed values and take over the values itself.

In both ways, the application is responsible for triggering the data transfer. There is no “secret” setting of data that is not under the control of your application.

Personalization Scenario, Personalization Scenario Sequence

Personalization scenarios and personalization scenario sequences are managed in the same way as they are used in the [Customized Layout](#) section.

73

Customized Proposals - Example

■ XML Layout	320
■ Java Adapter Code	321
■ Directly Accessing Proposal Values	322

The [address maintenance example](#) from the beginning of this part is now shown in detail. Since it is an extension of the previous example for the [customized layout](#), only the additions are shown that are responsible for the management of proposal values.

XML Layout

The XML layout definition is:

```
<page model="CustomizedLayoutAdapter">
  <titlebar name="Address Edit">
    <persedit persprop="paInfo">
      </persedit>
    </titlebar>
  <header withdistance="false">
    <button name="Save" method="onSave">
      </button>
    </header>
  <pagebody>
    <rowarea name="Address" visibleprop="adressAreaVisible">
      ...
      <itr visibleprop="townVisible">
        <label name="Town" width="100">
          </label>
        <field valueprop="town" width="200">
          </field>
        </itr>
      <itr visibleprop="countryVisible">
        <label name="Country" width="100">
          </label>
        <field valueprop="country" width="50">
          </field>
        </itr>
      ...
    </rowarea>
  </pagebody>
  <statusbar withdistance="false">
    </statusbar>
  <personalization>
    <persproposal property="town" comment="Town">
      </persproposal>
    <persproposal property="country" comment="Country">
      </persproposal>
    ...
    </persfilter>
  </personalization>
</page>
```

Below the PERSONALIZATION tag, you see two PERSPROPOSAL tags. Each tag holds the following information:

- The name of the property that should be proposed.
- A comment.

Java Adapter Code

The adapter code is:

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

public class CustomizedLayoutAdapter
    extends Adapter
{
    PersonalizationAdapterInfo m_paInfo = new PersonalizationAdapterInfo(this);
    ...

    ...

    public String getTown() { return m_town; }
    public void setTown(String value) { m_town = value; }

    public String getCountry() { return m_country; }
    public void setCountry(String value) { m_country = value; }
    ...
    public void init()
    {
        // switch maintenance on
        Personalization.switchPersonalizationMaintenanceOn(findSessionContext());
        // set up scenarios
        PersonalizationScenarioSequence pss = new ↵
PersonalizationScenarioSequence("customer");
        Personalization.defineScenarioSequenceInContext(findSessionContext(),pss);
        // transfer proposal values
        m_paInfo.applyProposals(this);
    }
    ...
}
```

You see that inside the `init()` method, the method `applyProposals(...)` is called. This method is responsible for transferring the proposal values into the corresponding properties of the adapter. For the transfer, just the normal methods are used: i.e. properties are either set via their corresponding set method or by calling the `IDynamicAccess` interface.

It is completely up to you where to embed the `applyProposals(...)` into your adapter code. To put it into the `init()` method is just one example. Maybe you want to make the decision whether to propose values or not dependent on some other conditions inside your program: when you create a new address, you want to propose values - however, when you edit an existing address, you do not want to propose values.

Directly Accessing Proposal Values

You can also access proposal values directly. The `PersonalizationAdapterInfo` class offers a corresponding method `getAllProposalValues()` to do so:

```
public void onDirectAccess()
{
    Properties props = m_paInfo.getAllProposalValues();
    if (props == null)
    {
        outputMessage(MT_ERROR, "No proposal values are available");
        return;
    }
    Iterator keys = props.keySet().iterator();
    while (keys.hasNext())
    {
        String key = (String)keys.next();
        String value = props.getProperty(key);
        System.out.println("Key/Value: " + key + "/" + value);
    }
}
```

XII

■ 74 Security Aspects	325
■ 75 Portal Integration	329
■ 76 Using Layout Painter Extensions	335
■ 77 Microsoft Silverlight Integration	353
■ 78 Integrating Application Designer Controls in HTML Pages	375
■ 79 Automated Testing	385

74

Security Aspects

With regard to AJAX, you have to keep in mind the following security risks:

1. Code Injection

The main risk is the so-called “code injection”.

To prevent code injection, all data needs to be checked on the server side. For this purpose, Application Designer offers the `IRequestDataConverter` interface. This filter enables you to convert values coming from the user interface client: each data request contains values of changed properties. Each property value that is transferred into the Application Designer server may be passed through an instance of this interface.

Background: in certain scenarios you may want to make sure that certain values are not passed into your application system. For example, for reasons of security, you do not want to enable inline scripting or inline SQL statements; therefore, you want to make sure that a user cannot input JavaScript statements or SQL statements. The *cisconfig.xml* file contains the parameter `requestdataconverter`. In order to make use of your data converter, specify the name of your data converter with this parameter. Example:

```
<cisconfig .. requestdataconverter="com.your.RequestDataConverter" />
```

2. Faked Client

The second risk is a so-called “faked client” which sends bad HTTP sequences, hoping that there is no server-side validation.

In this case, the responsibility is on the developer's side. Application Designer offers a client-side validation: the regular expression `/d/d/d/d` as the validation for a field; the field expects 4 decimal digits which will be validated on the client side. A faked client does not provide any validation. Thus, it depends on the developer to implement a server-side validation as well. One approach is to assume that all data that arrives from the client side might be wrong, even the data that is returned from a combo box. Therefore, if data might be wrong, it is important to double-check it on the server side.

3. Customizing Error Pages

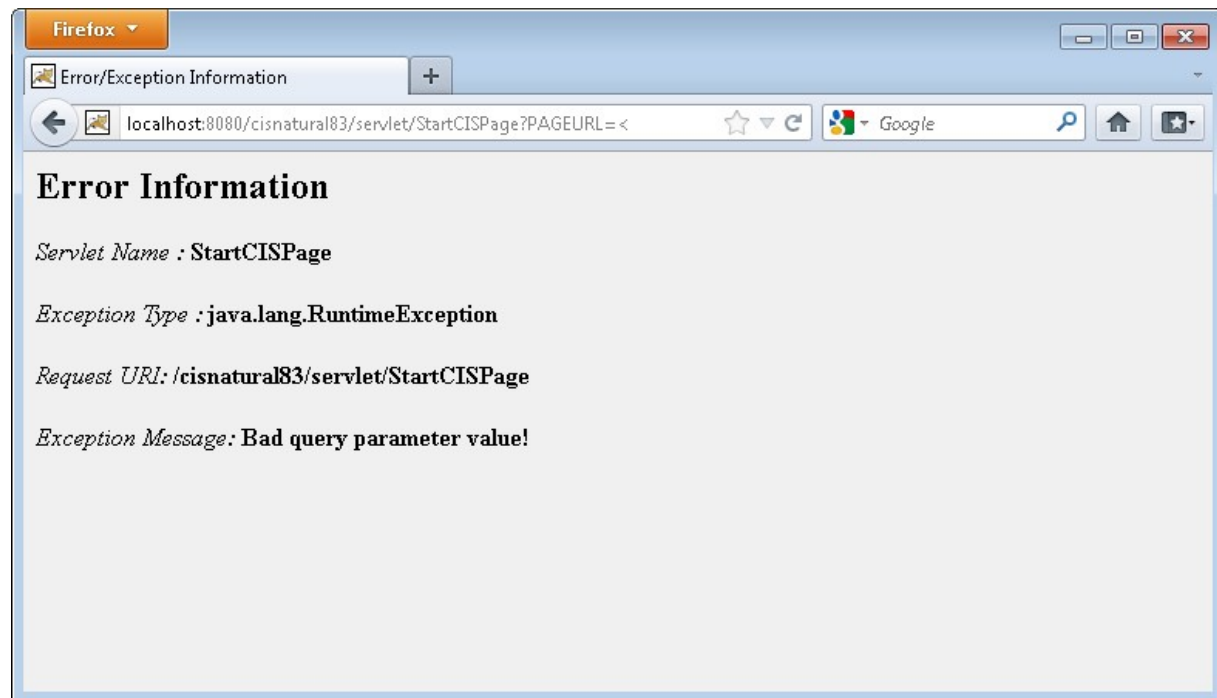
In the case of unexpected errors, most application servers show a default error page. This default error page mostly contains information such as stack traces. Showing full stack traces in a production environment is regarded as a security risk. To avoid this vulnerability, you can configure your own error pages in the *web.xml* file of your web application. For your convenience, the product contains a ready-to-use error handling servlet. The following example shows how to configure this servlet in the *web.xml* file:

```
<servlet id="DefaultErrorHandler">
    <servlet-name>DefaultErrorHandler</servlet-name>
    <servlet-class>com.softwareag.cis.server.DefaultErrorServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>DefaultErrorHandler</servlet-name>
    <url-pattern>/DefaultErrorHandler</url-pattern>
</servlet-mapping>

<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/DefaultErrorHandler</location>
</error-page>
```

The following is a sample error page that has been generated by the `com.softwareag.cis.server.DefaultErrorServlet`:



As an alternative to this default error handling servlet, you can add your own error handling servlets and/or error pages.

75

Portal Integration

■ Integrating Pages as Portlets	330
■ Session Management and Portlet API Support	332
■ Portlet Integration and AJAX	333

The Portlet API defines the way a portal server assembles pages out of content fragments that are provided by different applications. The Portlet API was published in its 1.0 version in October 2003 as JSR (Java Specification Request) 168. See the Java Community Process pages at <http://jcp.org/> for details. It is very important to point out the Portlet 1.0 Errata document which was published in May 2005 and which is an addition to the 1.0 standard. See <http://jcp.org/aboutJava/communityprocess/maintenance/jsr168/Portlet1.0-ERRATA.html> for details.

Application Designer provides an integration into portal servers that is based on the Portlet 1.0 standard. It allows you to easily import any page that is built using Application Designer into portal scenarios, i.e. the page can directly be used as part of a portal page.

The portal integration features are:

- Any page can be wrapped into a portal without coding effort.
- Parameters can be passed into the page by the normal URL extension (“&name=value”), by a POST or GET request.
- The application has full read and write access to the portlet request (and by this to the portlet session and to the portlet context) and thus can share and exchange data with other portlets.
- The deployment units (*.war* files) built by Application Designer are directly usable as input by portlet servers.

Integrating Pages as Portlets

There is one generic portlet that comes with Application Designer. The name is:

```
com.softwareag.cis.server.PortletWrapper
```

The portlet allows you to pass portlet preferences:

- PAGEURL - this is the name of the Application Designer page that is opened inside the portlet. The format is `"/project/page.html"`.
- NORMWIDTH/NORMHEIGHT - the width and height (as px/pt/% value) that the portlet should have when opened in *normal size*. The default is 400 for width and 300 for height.
- MAXWIDTH/MAXHEIGHT - the width and height (as px/pt/% value) that the portlet should have when opened in *maximum size*. The default is 100% both for width and height.

The specification of the portlet preferences needs to be done following the description of your portlet server.

Example: when deploying an Application Designer to the portlet reference implementation (Apache Pluto), then the portlets are registered in the *portletentityregistry.xml* file in the following way:

```

    <application id="8">
      <definition-id>cis</definition-id>
      <portlet id="1">
        <definition-id>cis.PortletWrapper</definition-id>
        <preferences>
          <pref-name>PAGEURL</pref-name>
          ↵
        <pref-value>/HTMLBasedGUI/com.softwareag.cis.admin.serverlog.html</pref-value>
          <read-only>true</read-only>
        </preferences>
        <preferences>
          <pref-name>MAXHEIGHT</pref-name>
          <pref-value>600</pref-value>
          <read-only>true</read-only>
        </preferences>
        <preferences>
          <pref-name>MAXWIDTH</pref-name>
          <pref-value>100%</pref-value>
          <read-only>true</read-only>
        </preferences>
      </portlet>
      <portlet id="2">
        <definition-id>cis.PortletWrapper</definition-id>
        <preferences>
          <pref-name>PAGEURL</pref-name>
          ↵
        <pref-value>/HTMLBasedGUI/com.softwareag.cis.workplace.logon.html</pref-value>
          <read-only>true</read-only>
        </preferences>
        <preferences>
          <pref-name>MAXHEIGHT</pref-name>
          <pref-value>600</pref-value>
          <read-only>true</read-only>
        </preferences>
        <preferences>
          <pref-name>MAXWIDTH</pref-name>
          <pref-value>100%</pref-value>
          <read-only>true</read-only>
        </preferences>
      </portlet>
      <portlet id="3">
        <definition-id>cis.PortletWrapper</definition-id>
        <preferences>
          <pref-name>PAGEURL</pref-name>
          <pref-value>/HTMLBasedGUI/xyz.html</pref-value>
          <read-only>true</read-only>
        </preferences>
        <preferences>
          <pref-name>MAXHEIGHT</pref-name>
          <pref-value>600</pref-value>
          <read-only>true</read-only>
        </preferences>
      </portlet>
    </application>

```

```
<preferences>
  <pref-name>MAXWIDTH</pref-name>
  <pref-value>100%</pref-value>
  <read-only>true</read-only>
</preferences>
</portlet>
</application>
```

The web application's name is *cis*. You see that per portlet a portlet definition is made, each definition pointing to the PAGEURL that it should open and each definition specifying the MAXHEIGHT and MAXWIDTH. In the Pluto environment the defined portlets can now be addressed from the portal page definition file (*pageregistry.xml*).

Session Management and Portlet API Support

The portal server is responsible for a certain session management (and e.g. for single signon support within this session management).

Application Designer comes with its own session management. The coupling is done in the following way: the portal's session management is seen as the "Master Session Management". This means that sessions are opened inside Application Designer at the point of time when they are first accessed through a portlet. The session ID that is used inside Application Designer is the same as the session ID that comes from the portal server. The session's info string is set to "Created by Portlet Wrapper".

In Application Designer, the server side counterpart of a page that runs inside the browser is also called "adapter object".

From the adapter you can directly access the portlet request which comes from the portal server:

- The method `PortletWrapper.findPortletRequest(this)` returns an object of type `PortletRequest`.

From the portlet request you can navigate to

- the portlet session
- the portlet credential information
- the portlet context

following the Portlet 1.0 specification.

Portlet Integration and AJAX

AJAX technology is used inside Application Designer to ensure that pages in the browser are not permanently fully reloaded when a page communicates to its server side application.

Portal pages take a different approach: portal pages are always fully reloaded (fully means: all portlets that are rendered inside one page are reloaded) whenever one portlet communicates to its server side application.

The portlet integration of Application Designer takes this approach into account:

- Whenever the Application Designer portlet exchanges information with its server side counterpart, NO reloading of the whole portal page is done internally. You can imagine the portlet page to be an "island" whose communication is decoupled from the portal server's page updating.

This ensures that complex pages requiring AJAX for supporting a high frontend interactivity are also usable within a portal environment.

76

Using Layout Painter Extensions

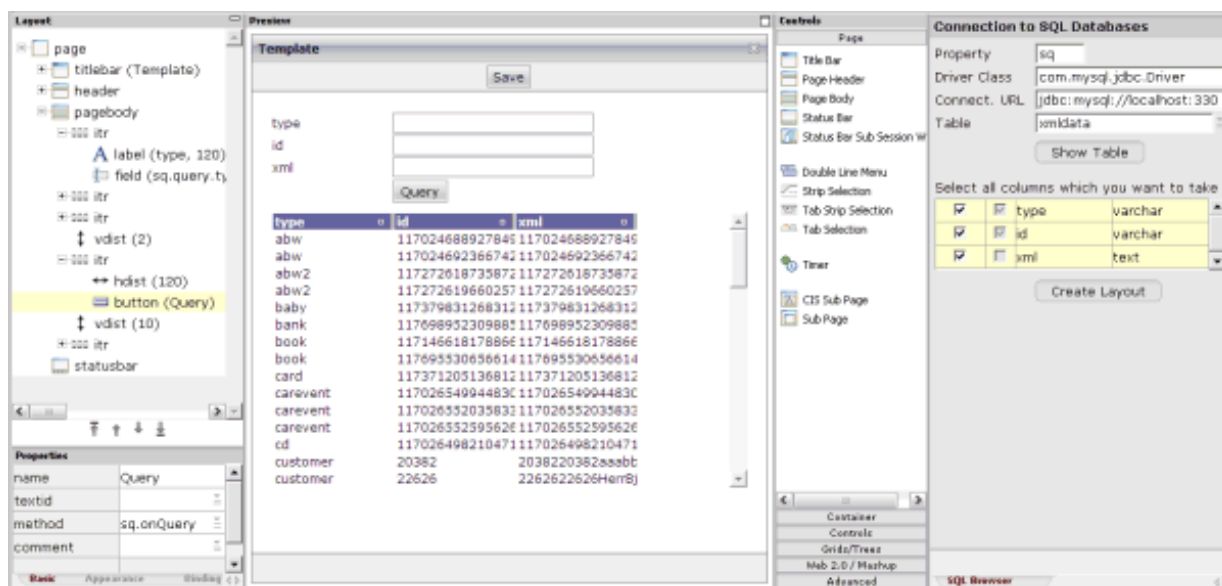
■ Example	336
■ Details on the Extension	337
■ Extension Meets Pattern	347

You can place external tools into a dedicated area of the Layout Painter and get access to the XML layout that is currently edited. Thus, you can build editor extensions which typically generate a certain part of an XML layout which can then be added to the XML layout that is currently edited in the Layout Painter.

For information on how to use the Layout Painter, see *Layout Painter* in the *Development Workplace* documentation.

Example

Using extensions, you can easily generate layout elements into an existing layout. This example shows how to build SQL query screens. The result will look as follows:



The right side of the Layout Painter provides an additional extension area in which you can enter database parameters and in which the columns of a selected table are shown. This extension is used in the following way:

- You specify a property name. This property name will later occur in the names of the properties for the generated layout elements.

Example: if you specify "sq" as the property name, the query field **type** may point, for example, to the property `sq.query.type`.

- You specify the class name and the connection URL of the database to be accessed.



Note: There are also other ways of accessing a database, however, this example concentrates on Layout Painter extension concepts, not on database management.

- The column definitions of the table are shown. You select the columns that are to appear in the layout and choose the **Create Layout** button. As a result, a layout is generated into the page body that reflects a query screen.
- You can manipulate the query screen using the normal edit functionality of the Layout Painter.

Details on the Extension

The definition of an extension is simple:

- You define the extension screen and adapter just as you define a normal Application Designer page.
- You add the interface `IEditorExtension` to the adapter implementation.
- You register the extension in the *cisconfig.xml* file.

In the above example, the layout of the extension is defined in the following way:

```
<page model="com.softwareag.cis.editor.sql.SQLExtensionAdapter">
  <titlebar name="Connection to SQL Databases" withclose="false">
    </titlebar>
  <pagebody horizdist="false" pagebodystyle="background-color: #deebf7" ↵
paddingleft="5"
      paddingright="5" paddingtop="5" paddingbottom="5">
    <itr>
      <label name="Property" width="100">
        </label>
      <field valueprop="prefix" width="50">
        </field>
      </itr>
    <itr takefullwidth="true" fixlayout="true">
      <label name="Driver Class" width="100">
        </label>
      <field valueprop="driverClass" width="100%">
        </field>
      </itr>
    <itr takefullwidth="true" fixlayout="true">
      <label name="Connect. URL" width="100">
        </label>
      <field valueprop="connectionURL" width="100%">
        </field>
      </itr>
    <itr takefullwidth="true">
      <label name="Table" width="100">
        </label>
      <field valueprop="table" width="100%" ↵
      popupmethod="openIdValueComboOrPopup">
        </field>
      </itr>
  </pagebody>
</page>
```

```

    </itr>
    <vdist height="5">
    </vdist>
    <itr>
        <hdist width="100">
        </hdist>
        <button name="Show Table" method="onShowTable">
        </button>
    </itr>
    <vdist height="15">
    </vdist>
    <rowdynavis valueprop="tableVisible">
        <itr takefullwidth="true">
            <label name="Select all columns which you want to take over into
your layout. Then press &quot;Create Layout&quot;." asplaintext="true">
            </label>
        </itr>
        <rowtablearea2 griddataprop="columns" rowcount="10" vscroll="auto"
            firstrowcolwidths="true">
            <repeat>
                <str valueprop="selected" showifempty="false">
                    <checkbox valueprop="selected" flush="server" width="50"
align="center">
                    </checkbox>
                    <checkbox valueprop="key" width="30" displayonly="true">
                    </checkbox>
                    <textout valueprop="column" width="50%">
                    </textout>
                    <textout valueprop="type" width="50%">
                    </textout>
                </str>
            </repeat>
        </rowtablearea2>
        <vdist height="10">
        </vdist>
        <itr>
            <hdist width="100">
            </hdist>
            <button name="Create Layout" method="onCreateLayout">
            </button>
        </itr>
    </rowdynavis>
</pagebody>
<statusbar withdistance="false">
</statusbar>
</page>

```

The adapter implementation looks as follows:

```

package com.softwareag.cis.editor.sql;

// not shown: ...package import statements...

public class SQLExtensionAdapter
    extends Adapter
    implements IEditorExtension
{
    // -----
    // inner classes
    // -----

    public class ColumnsItem
    {
        String m_column;
        public String getColumn() { return m_column; }
        public void setColumn(String value) { m_column = value; }

        boolean m_key;
        public boolean getKey() { return m_key; }
        public void setKey(boolean value) { m_key = value; }

        boolean m_selected;
        public boolean getSelected() { return m_selected; }
        public void setSelected(boolean value) { m_selected = value; }

        String m_type;
        public String getType() { return m_type; }
        public void setType(String value) { m_type = value; }
    }

    GRIDCollection m_columns = new GRIDCollection();
    public GRIDCollection getColumns() { return m_columns; }

    String m_connectionURL;
    public String getConnectionURL() { return m_connectionURL; }
    public void setConnectionURL(String value) { m_connectionURL = value; }

    String m_driverClass;
    public String getDriverClass() { return m_driverClass; }
    public void setDriverClass(String value) { m_driverClass = value; }

    String m_prefix;
    public String getPrefix() { return m_prefix; }
    public void setPrefix(String value) { m_prefix = value; }

    String m_table;
    public String getTable() { return m_table; }
    public void setTable(String value) { m_table = value; }

    boolean m_tableVisible = false;
    public boolean getTableVisible() { return m_tableVisible; }

```

```
IExtensionPoint m_extensionPoint;

// -----
// IEditorExtension
// -----

public String buildPageURL()
{
    return "/HTMLBasedGUI/cis.editor.SQLExtension";
}

public void init(IExtensionPoint extensionPoint)
{
    m_extensionPoint = extensionPoint;
}

public void reactOnLoadDocument() {}
public void reactOnUnloadDocument() {}

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

public void init()
{
    m_driverClass = "com.mysql.jdbc.Driver";
    m_connectionURL = ↵
"jdbc:mysql://localhost:3306/cispersist?user=root&password=admin";
}

/** */
public ValidValueLine[] findValidValuesForTable()
{
    return SQLUtil.findValidValuesForTable(m_driverClass,m_connectionURL);
}

/**
 * Read table meta data and transfer into columns list.
 */
public void onShowTable()
{
    if (m_prefix == null || m_prefix.trim().length() == 0)
    {
        outputMessage(MT_ERROR,"Please specify property first");
        return;
    }
    if (m_table == null || m_table.trim().length() == 0)
    {
        outputMessage(MT_ERROR,"Please specify table name first");
        return;
    }
}
```

```

        // fetch meta data from SQL and load into columns table
        try
        {
            SQLUtil.ColumnMetaData[] cols = ↵
SQLUtil.readMetaDataForTable(m_driverClass,m_connectionURL,m_table);
            m_columns.clear();
            for (int i=0; i<cols.length; i++)
            {
                ColumnsItem ci = new ColumnsItem();
                ci.setColumn(cols[i].m_column);
                ci.setType(cols[i].m_type);
                ci.setKey(cols[i].m_isKey);
                m_columns.add(ci);
            }
            m_tableVisible = true;
        }
        catch (Throwable t)
        {
            outputMessage(MT_ERROR,t.toString());
        }
    }

    /**
     * Create the layout out of the column definition and transfer the layout
     * into the page's XML.
     */
    public void onCreateLayout()
    {
        List selColumns = new ArrayList();
        Iterator iter = m_columns.iterator();
        while (iter.hasNext())
        {
            ColumnsItem ci = (ColumnsItem)iter.next();
            if (ci.getSelected() == true)
                selColumns.add(ci);
        }
        if (selColumns.size() == 0)
        {
            outputMessage(MT_ERROR,"Please select all columns for which layout ↵
elements should be generated");
            return;
        }
        // generate XML layout definition
        StringBuffer sb = new StringBuffer();
        iter = selColumns.iterator();
        while (iter.hasNext())
        {
            ColumnsItem ci = (ColumnsItem)iter.next();
            sb.append("<itr>");
            sb.append("<label name='"+ci.getColumn()+"' width='120'/>");
            sb.append("<field valueprop='"+m_prefix+".query."+ci.getColumn()+"' ↵
width='200'/>");

```

```
        sb.append("</itr>");
    }
    sb.append("<vdist height='2'/>");
    sb.append("<itr>");
    sb.append( "<hdist width='120'/>");
    sb.append( "<button name='Query' method='"+m_prefix+".onQuery'/>");
    sb.append("</itr>");
    sb.append("<vdist height='10'/>");
    sb.append("<itr takefullwidth='true'>");
    sb.append( "<textgridsss2 width='100%' rowcount='15' ↵
griddataprop='"+m_prefix+".lines'>");
    iter = selColumns.iterator();
    while (iter.hasNext())
    {
        ColumnsItem ci = (ColumnsItem)iter.next();
        sb.append("<column name='"+ci.getColumn()+"' ↵
property='"+ci.getColumn()+"' width='120'/>");
    }
    sb.append( "</textgridsss2>");
    sb.append("</itr>");
    // pass created XML into the page
    if (m_extensionPoint != null)
    {
        String layoutXML = m_extensionPoint.findXMLLayout();
        layoutXML = ↵
StringMgmt.replaceInString("</pagebody>",sb.toString()+"</pagebody>",layoutXML);
        m_extensionPoint.updateXMLLayout(layoutXML);
    }
}
}
```

The building blocks of the code are:

- The adapter implements the interface `IEditorExtension` and therefore needs to implement the following methods:

- `buildPageURL`

This method tells the extension framework the name of the extension's HTML page. The extension is registered at the server side. Therefore, the Layout Painter needs to know which page it has to embed in the extension area.

- `init`

This method is called by the Layout Painter. The most important parameter is the `IExtensionPoint` which is an abstraction of the Layout Painter environment. It allows access to the inner parts of the Layout Painter such as the XML layout definition.

- `reactOnLoadDocument` **and** `reactOnUnloadDocument`
These methods are called when the user opens or closes a layout definition inside the editor.
- The adapter provides a method `onShowTable()` which accesses the database via an `SQLUtil` class and fills the grid containing information on each column.
- The adapter provides a method `onCreateLayout()`. This method creates a layout which is then passed into the page's layout. For the generation of the layout, the following rules apply:
 - For each column that is selected, a query line is generated. The query line is an ITR line holding a LABEL and a FIELD definition.
 - One `TEXTGRIDSSS2` definition is created. For each selected column, one `COLUMN` definition inside the grid is created.

The `SQLUtil` class provides generic functions for accessing the database:

```
package com.softwareag.cis.editor.sql;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.ServerLog;
import com.softwareag.cis.server.util.ValidValueLine;
import com.softwareag.cis.editor.sql.SQLExtensionAdapter.ColumnsItem;

/**
 * Collection of static functions which manage the interface
 * to the database.
 */
public class SQLUtil
{
    // -----
    // inner classes
    // -----

    public static class ColumnMetaData
    {
        public boolean m_isKey;
        public String m_column;
        public String m_type;
    }
}
```

```
// -----  
// public usage  
// -----  
  
/**  
 */  
public static Connection createConnection(String driverClassName,  
                                         String connectionURL)  
    throws Exception  
{  
    Class.forName(driverClassName).newInstance();  
    Connection conn = DriverManager.getConnection(connectionURL);  
    return conn;  
}  
  
/**  
 */  
public static ColumnMetaData[] readMetaDataForTable(String driverClassName,  
                                                    String connectionURL,  
                                                    String table)  
{  
    try  
    {  
        List columns = new ArrayList();  
        Connection conn = createConnection(driverClassName,connectionURL);  
        DatabaseMetaData dbmd = conn.getMetaData();  
        // get primary key and column info  
        Set primaryKeys = new HashSet();  
        ResultSet rs = dbmd.getPrimaryKeys(null,null,table);  
        while (rs.next())  
            primaryKeys.add(rs.getString("COLUMN_NAME"));  
        rs = dbmd.getColumns(null,null,table,null);  
        while (rs.next())  
        {  
            ColumnMetaData cmd = new ColumnMetaData();  
            cmd.m_column = rs.getString("COLUMN_NAME");  
            cmd.m_type = rs.getString("TYPE_NAME");  
            if (primaryKeys.contains(cmd.m_column))  
                cmd.m_isKey = true;  
            columns.add(cmd);  
        }  
        conn.close();  
        ColumnMetaData[] result = new ColumnMetaData[columns.size()];  
        columns.toArray(result);  
        return result;  
    }  
    catch (Throwable t)  
    {  
        throw new Error(t);  
    }  
}
```

```

/**
 */
public static ValidValueLine[] findValidValuesForTable(String driverClassName,
                                                    String connectionURL)
{
    try
    {
        Connection conn = createConnection(driverClassName,connectionURL);
        DatabaseMetaData dbmd = conn.getMetaData();
        // get primary key and column info
        List vvs = new ArrayList();
        ResultSet rs = dbmd.getTables(null,null,null,null);
        while (rs.next())
        {
            ValidValueLine vv = new ValidValueLine(rs.getString("TABLE_NAME"));
            vvs.add(vv);
        }
        conn.close();
        ValidValueLine[] result = new ValidValueLine[vvs.size()];
        vvs.toArray(result);
        return result;
    }
    catch (Throwable t)
    {
        ServerLog.appendException(t);
        return new ValidValueLine[0];
    }
}

/**
 */
public static List executeQuery(String driverClassName,
                               String connectionURL,
                               String sql,
                               String[] columns)
{
    try
    {
        List resultList = new ArrayList();
        Connection conn = createConnection(driverClassName,connectionURL);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        while (rs.next())
        {
            Map columnMap = new HashMap();
            for (int i=0; i<columns.length; i++)
            {
                String column = columns[i];
                String value = rs.getString(column);
                columnMap.put(column,value);
            }
        }
    }
}

```

```

        resultList.add(columnMap);
    }
    conn.close();
    return resultList;
}
catch (Throwable t)
{
    return new ArrayList();
}
}
}

```



Note: The functions for table processing are a bit limited as they deal only with string-type columns, however, this is a demo use case only.

Finally, let us have a look at the *cisconfig.xml* file in which all extensions are registered. For this example, we have inserted a new editor extension named "SQL Browser".

```

<cisconfig startmonitoringthread="true"
    requestclienthost="false"
    debugmode="false"
    loglevel="EWI"
    logtoscreen="true"
    sessiontimeout="3600"
    xmldatamanager="com.softwareag.cis.xmldata.filebased.XMLDataManager"
    useownclassloader="true"
    browserpopuponerror="false"
    framebuffersize="3"
    ↵
onlinehelpmanager="com.softwareag.cis.onlinehelp.projectbased.FrameHelpOHManager"
    textencoding="UTF-8"
    enableadapterpreload="true"
    animatecontrols="true">

    <requestrecording recordrequests="false"
        recorddirectory="c:/temp/traces/">
    </requestrecording>

    <editorextensions>
        <editorextension name="SQL Browser"
            classname="com.softwareag.cis.editor.sql.SQLExtensionAdapter">
        </editorextension>
        <editorextension name="Map Converter" classname="PluginMapCreatorAdapter">
        </editorextension>
    </editorextensions>

    <generationaddons>
        <generationaddon ↵
classname="com.softwareag.cis.gui.generate.XSDGenerationAddon">
        </generationaddon>

```

```
</generationaddons>
</cisconfig>
```

In the section `editorextensions`, the class names of all editor extensions are listed. The class needs to provide a constructor without parameters and needs to implement the interface `IEditorExtension`.

Extension Meets Pattern

A Layout Painter extension generates a certain XML layout which is taken over into the page. It makes sense to generate the layout in such a way that it meets a processing pattern within the adapter of the generated page.

In the above SQL example, the controls that are generated by the extension are binding to properties in the following way:

- There is a general property definition that is input into the extension. Let us assume that the user specifies "sq" as the general property name.
- All query fields are binding to a property definition `valueprop="sq.query.columnName"`.
- The `TEXTGRIDSSS2` binds to `griddataprop="sq.lines"` and each `COLUMN` definition inside the grids binds to `property="columnName"`.

Let us have a look at the adapter code of the page that is generated inside the Layout Painter:

```
// 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.*;
import com.softwareag.cis.editor.sql.SQLQueryMgr;

public class SSQQLL1Adapter
    extends Adapter
{
    SQLQueryMgr m_sq = new SQLQueryMgr(
        "com.mysql.jdbc.Driver",
        "jdbc:mysql://localhost:3306/cispersist?user=root&password=admin",
        "xmldata");
    public SQLQueryMgr getSq() { return m_sq; }
}
```

This is not much code. The main coding is done in the `SQLQueryMgr` class. This pattern class can be used throughout various adapters to provide the ".query" and ".lines" data and processing. Let us have a look at this class:

```
package com.softwareag.cis.editor.sql;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import com.softwareag.cis.file.CSVManager;
import com.softwareag.cis.server.IDynamicAccess;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

public class SQLQueryMgr
{
    // -----
    // inner classes
    // -----

    /**
     * This is the object that holds all the query parameters. Each column
     * is represented as one property which is reachable by IDynamicAccess.
     */
    public class TableRow
        implements IDynamicAccess
    {
        Map m_data = new HashMap();
        public String[] findDynamicAccessProperties()
        {
            return m_columnNames;
        }
        public Class getClassForProperty(String property)
        {
            return null; // default: String...
        }
        public Object getPropertyValue(String propertyName)
        {
            return this.m_data.get(propertyName);
        }
        public void invokeMethod(String methodName)
        {
        }
        public void setPropertyValue(String propertyName, Object value)
        {
            this.m_data.put(propertyName,value);
        }
    }

    public class SelectableTableRow
        extends TableRow
    {
        boolean m_selected;
    }
}
```

```

        public void setSelected(boolean value) { this.m_selected = value; }
        public boolean getSelected() { return this.m_selected; }
    }

    // -----
    // members
    // -----

    String m_driverClassName;
    String m_connectionURL;
    String m_table;
    SQLUtil.ColumnMetaData[] m_columns; // loaded in init()
    String[] m_columnNames; // loaded in init()

    TableRow m_query = new TableRow();
    TEXTGRIDCollection m_lines = new TEXTGRIDCollection();

    // -----
    // constructors
    // -----

    public SQLQueryMgr(String driverClassName,
                      String connectionURL,
                      String table)
    {
        m_driverClassName = driverClassName;
        m_connectionURL = connectionURL;
        m_table = table;
        m_columns =
SQLUtil.readMetaDataForTable(m_driverClassName,m_connectionURL,m_table);
        m_columnNames = new String[m_columns.length];
        for (int i=0; i<m_columnNames.length; i++)
            m_columnNames[i] = m_columns[i].m_column;
    }

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

    public TableRow getQuery() { return m_query; }
    public TEXTGRIDCollection getLines() { return m_lines; }

    public void onQuery()
    {
        try
        {
            // build SQL string
            StringBuffer sql = new StringBuffer();
            sql.append("SELECT * FROM " + m_table + " ");
            int counter = 0;
            for (int i=0; i<m_columnNames.length; i++)
            {

```

```
        String colName = m_columnNames[i];
        Object colValue = m_query.getPropertyValue(colName);
        if (colValue == null) continue;
        if (counter == 0)
            sql.append("WHERE ");
        else
            sql.append("AND ");
        sql.append(colName + " LIKE " + "'" + colValue + "'");
        counter++;
    }
    sql.append(";");
    // execute query
    List queryList = SQLUtil.executeQuery(m_driverClassName,
                                         m_connectionURL,
                                         sql.toString(),
                                         m_columnNames);

    // transfer into lines
    m_lines.clear();
    Iterator iter = queryList.iterator();
    while (iter.hasNext())
    {
        Map columnMap = (Map)iter.next();
        SelectableTableRow str = new SelectableTableRow();
        for (int i=0; i<m_columnNames.length; i++)
        {
            ↵
            str.setPropertyValue(m_columnNames[i],columnMap.get(m_columnNames[i]));
            m_lines.add(str);
        }
    }
    catch (Throwable t)
    {
        t.printStackTrace();
    }
}
```

The building blocks are:

- There is a class `TableRow` which is used both for the query properties (`sq.query.type`, `sq.query.id`, etc.) and for the line objects of the grid. This class supports a dynamic set of properties using the `IDynamicAccess` interface. The definition which properties to support comes from the table definition. Again, the `SQLUtil` class is used for accessing the table definition.
- There is an `onQuery` method which builds the SQL selection string out of the query parameters, passes it to the SQL database processing and transfers the result back into the grid collection (`m_lines`).

As you have seen in the above example, only a small amount of coding is required for building extensions that greatly simplify the creation of typical screens. The XML layout that is added using

extensions typically matches a pattern on the server side that provides the properties and the processing “without coding” (as seen from the usage point of view).

77

Microsoft Silverlight Integration

■ Example	354
■ Implementation of the Sample Page	355
■ Integration of Silverlight	358

Microsoft Silverlight is a plug-in which allows to specify sophisticated graphics, such as animations. It is positioned as competitive plug-in to Adobe's Macromedia Flash control. Silverlight runs within Microsoft-based and Apple-OS-based systems. As browsers, the Internet Explorer, Mozilla and Safari are supported.

This chapter assumes that you already know how to deal with Silverlight. Read the documentation that comes with the Silverlight SDK when downloading from Microsoft.

The integration of Silverlight is simple and straight forward. Silverlight has direct access to JavaScript processing - this is where AJAX and Silverlight meet.

Using Application Designer's control concept, the Silverlight part of a page can be nicely integrated, allowing the AJAX page to pass data ("net data") to the Silverlight control and allowing the Silverlight processing to set data and raise events within the AJAX processing.

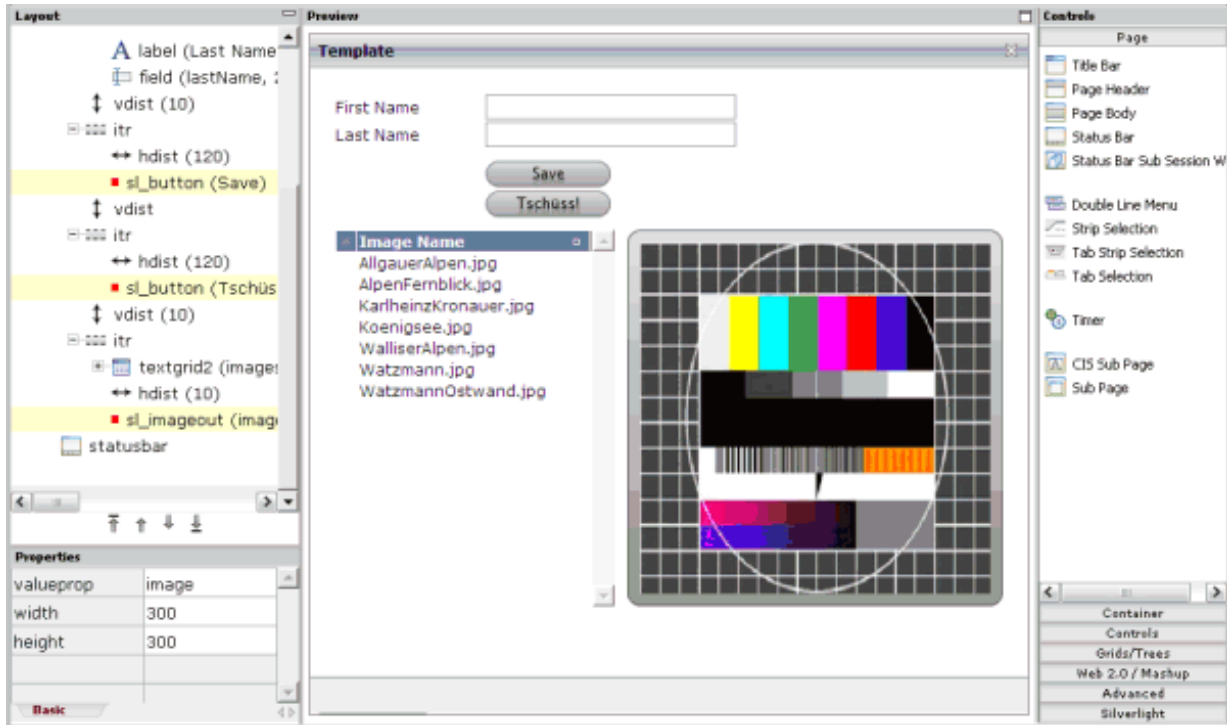


Notes:

1. Silverlight is bound to certain platforms (Windows, Mac OS). It is up to you to decide whether you should use it or not.
2. Silverlight is browser-specific, the main reason being the communication between the Silverlight plug-in and the surrounding environment. In the browser, the binding is done using JavaScript.
3. The Silverlight plug-in only supports a subset of Microsoft's XAML definitions. The subset definition is documented as part of the SDK that you can download from Microsoft.

Example

The following page contains three special controls: two buttons and an image area.



As you can see in the layout tree, the names of the corresponding control tags are SL_BUTTON and SL_IMAGEOUT. These controls are animated:

- When pressing a button, it rotates one time.
- When loading images into the image area, the images fly in and out.

From the layout perspective, the controls are just normal controls. The button is bound to a server-side method, the image output area is bound to a server-side property. The control's complexity is completely hidden from the application developer who just uses controls by dropping them into a page and putting server-side logic behind them.

Implementation of the Sample Page

The layout of the above sample page is defined in the following way:

```
<page model="SL_Test1Adapter">
  <sl_silverlight>
  </sl_silverlight>
  <titlebar name="Template">
  </titlebar>
  <pagebody paddingleft="20" paddingright="20" paddingtop="20" paddingbottom="20">
    <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>
    <vdist height="10">
    </vdist>
    <itr>
        <hdist width="120">
        </hdist>
        <sl_button name="Save" method="onSave">
        </sl_button>
    </itr>
    <vdist>
    </vdist>
    <itr>
        <hdist width="120">
        </hdist>
        <sl_button name="Tschüss!" method="onBye">
        </sl_button>
    </itr>
    <vdist height="10">
    </vdist>
    <itr takefullwidth="true">
        <textgrid2 griddataprop="images" width="100%" height="100%" ↵
selectprop="selected" singleselect="true" onclickmethod="onSelectImage">
            <column name="Image Name" property="imageName" width="100%">
            </column>
        </textgrid2>
        <hdist width="10">
        </hdist>
        <sl_imageout valueprop="image" width="300" height="300">
        </sl_imageout>
    </itr>
</pagebody>
<statusbar withdistance="false">
</statusbar>
</page>

```

In the above layout, the following types of tags are used that are specific to Silverlight. These are custom controls.

■ SL_SILVERLIGHT

A tag that puts some Silverlight initialization code into the page. This control has no visible representation in the page; it just adds some functions which are required for the visible Silverlight controls.

■ SL_BUTTON

The button tag.

■ SL_IMAGEOUT

The image output tag.

The adapter implementation looks as follows:

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

import java.util.*;

import com.softwareag.cis.file.FileManager;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class SL_Test1Adapter
    extends Adapter
{
    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; }

    String m_image;
    public String getImage() { return m_image; }
    public void setImage(String value) { m_image = value; }

    public class ImagesItem
    {
        // property >imageName<
        String m_imageName;
        public String getImageName() { return m_imageName; }
        public void setImageName(String value) { m_imageName = value; }

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

    TEXTGRIDCollection m_images = new TEXTGRIDCollection();
    public TEXTGRIDCollection getImages() { return m_images; }

    public void init()
    {
        String dirName = Params.getApplicationDirectoryName("playground") + "images/";
        String[] images = FileManager.getFileNamesOfDirectory(dirName, ".jpg");
    }
}
```

```
        for (int i=0; i<images.length; i++)
        {
            ImagesItem ii = new ImagesItem();
            ii.setImageName(images[i]);
            m_images.add(ii);
        }
    }

    public void onSelectImage()
    {
        ImagesItem ii = (ImagesItem)m_images.findLastSelectedItem();
        m_image = "../playground/images/"+ii.getImageName();
    }

    public void onBye()
    {
        outputMessage(MT_SUCCESS, "Bye was pressed!");
    }

    public void onSave()
    {
        outputMessage(MT_SUCCESS, "Save was pressed!");
    }
}
```

The adapter code does not contain any items which are specific to Silverlight. The important items in the code are:

- The text grid is loaded in the `init()` method. A certain directory is scanned for JPG files and a table row is created for each item.
- Once the user selects a text grid item, the `onSelect()` method takes care of transferring the name of the selected image into the property `m_image`. This is the property which is referenced by the `SL_IMAGEOUT` control.

Integration of Silverlight

The descriptions below are based on the following structure:

```
/HTMLBasedGUI/
    /silverlight/
        /createSilverlight.js
        /Silverlight.js
        /SL_BUTTON.xaml
        /SL_IMAGEOUT.xaml
```

There is an `/HTMLBasedGUI/silverlight` directory in which the following files are kept:

■ *createSilverlight.js*

This file contains JavaScript that generated the HTML elements (for example, an OBJECT tag) that are holding a Silverlight plug-in. This file was taken from the Silverlight SDK without any change.

■ *Silverlight.js*

This file contains binding code between the JavaScript processing of the page and the Silverlight plug-ins.

■ **.xaml*

The **.xaml* files (*SL_BUTTON.xaml* and *SL_IMAGEOUT.xaml*) hold the XAML rendering definition for the Silverlight controls.

This example introduces new control handlers for the SL_* controls .



Note: The selection of the *HTMLBasedGUI* directory for keeping the *silverlight* directory was done because *HTMLBasedGUI* is can be considered as Software AG's "home directory" within a web application. When doing your own implementation, you can choose any directory of your choice that is part of the web application's directory.

The custom controls are described below:

- [SL_SILVERLIGHT Control](#)
- [SL_BUTTON](#)
- [SL_IMAGEOUT](#)

SL_SILVERLIGHT Control

Editor Extension File

Let us first have a look at the editor extension file that defines the control's integration into the Layout Painter.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

    <!--
    *****
    * TAGs
    *****
    -->

    <!--SILVERLIGHT -->
```

```
<tag name="silverlight">
</tag>
<tagsubnodeextension control="page" newsubnode="silverlight"/>

</editor>
</controllibrary>
```

The definition is pretty simple. The tag will appear in the controls palette of the Layout Painter, as a new section with the name "Silverlight". The control does not have any properties.

Control Handler

The control adds some JavaScript statements into the generated HTML of a page. Have a look at the implementation for the control handler:

```
package com.softwareag.cis.gui.generate;

import org.xml.sax.AttributeList;

import com.softwareag.cis.gui.protocol.ProtocolItem;

public class SL_SILVERLIGHTHandler
    implements ITagHandler
{
    // -----
    // public usage
    // -----

    public void generateHTMLForStartTag(int id,
                                       String tagName,
                                       AttributeList attrlist,
                                       ITagHandler[] handlersAbove,
                                       StringBuffer sb,
                                       ProtocolItem protocolItem)
    {
        sb.append("<!-- SILVERLIGHT begin -->\n");
        sb.append("<script type=\"text/javascript\" ↵
src=\"../HTMLBasedGUI/silverlight/Silverlight.js\"></script>\n");
        sb.append("<script type=\"text/javascript\" ↵
src=\"../HTMLBasedGUI/silverlight/createSilverlight.js\"></script>\n");
    }

    public void generateHTMLForEndTag(String tagName, StringBuffer sb)
    {
        sb.append("<!-- SILVERLIGHT end -->\n");
    }

    public void generateJavaScriptForInit(int id,
                                       String tagName,
                                       StringBuffer sb)
    {

```

```
}
}
}
```

This implementation only contains the integration of the *.js* files which are stored in the */HTML-BasedGUI/silverlight* directory.

SL_BUTTON

The handling of the button is done in several parts.

Editor Extension File

Let us first have a look at the editor extension file that defines the control's integration in the Layout Painter.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

    <!--
    *****
    * TAGs
    *****
    -->

    <!-- SL_BUTTON -->
    <tag name="sl_button">
      <attribute name="name"/>
      <attribute name="method"/>
      <attribute name="width" datatype="width"/>
      <attribute name="height" datatype="height"/>
    </tag>
    <tagsubnodeextension control="itr" newsubnode="sl_button"/>
    <tagsubnodeextension control="tr" newsubnode="sl_button"/>
    <taggroupsubnodeextension group="Silverlight" newsubnode="sl_button"/>

  </editor>
</controllibrary>
```

The control has four attributes. *name*, *width* and *height* are used to specify the button's rendering. *method* defines the binding to an adapter method.

In the controls palette of the Layout Painter, the control will appear within the section **Silverlight**. It can be placed below ITR and TR row containers.

Control Handler

The SL_BUTTON tag handler class is responsible for generating the HTML:

```
package com.softwareag.cis.gui.generate;

import org.xml.sax.AttributeList;

import com.softwareag.cis.gui.protocol.Message;
import com.softwareag.cis.gui.protocol.ProtocolItem;

public class SL_BUTTONHandler
    implements ITagHandler
{
    // -----
    // members
    // -----

    String m_name = "";
    String m_method;
    String m_width = "100";
    String m_height= "21";

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

    public void generateHTMLForStartTag(int id,
                                        String tagName,
                                        AttributeList attrlist,
                                        ITagHandler[] handlersAbove,
                                        StringBuffer sb,
                                        ProtocolItem protocolItem)
    {
        readAttributes(attrlist);
        fillProtocol(protocolItem);
        generateXSD(protocolItem);
        // create HTML
        sb.append("<!-- SL_BUTTON begin -->\n");
        sb.append("<td width='"+m_width+"'><div id='C"+id+"'"+
                  " style='width:"+m_width+"; height:"+m_height+"'></td>\n");
        sb.append("<script>function onButtonPressed"+id+"()"+
                  " { C.invokeMethodInModel('"+m_method+"'); }</script>");
    }

    public void generateHTMLForEndTag(String tagName, StringBuffer sb)
    {
        sb.append("<!-- SL_BUTTON end -->\n");
    }

    public void generateJavaScriptForInit(int id,
```

```

        String tagName,
        StringBuffer sb)
    {
        sb.append("if (firstusage) C_"+id+" = document.getElementById('C_"+id+"');\n");
        sb.append("if (firstusage) ↵
createMySilverlightControl('../HTMLBasedGUI/silverlight/SL_BUTTON.xaml',C_"+id+", '"+id+"', '"+m_width+"', '"+m_height+"');\n");
        sb.append("if (firstusage) C_"+id+".CASA_width='"+m_width+"';\n");
        sb.append("if (firstusage) C_"+id+".CASA_method='"+m_method+"';\n");
        sb.append("if (firstusage) C_"+id+".CASA_name='"+m_name+"';\n");
    }

    // -----
    // private usage
    // -----

    /** */
    private void readAttributes(AttributeList attrlist)
    {
        for (int i=0; i<attrlist.getLength(); i++)
        {
            if (attrlist.getName(i).equals("name")) m_name = attrlist.getValue(i);
            if (attrlist.getName(i).equals("method")) m_method = attrlist.getValue(i);
            if (attrlist.getName(i).equals("width")) m_width = attrlist.getValue(i);
            if (attrlist.getName(i).equals("height")) m_height = attrlist.getValue(i);
        }
    }

    /** */
    private void fillProtocol(ProtocolItem pi)
    {
        pi.addMethod(m_method);
    }

    /** */
    private void generateXSD(ProtocolItem pi)
    {
        XSDGenerationAddon xga = ↵
(XSDGenerationAddon)pi.findGenerationAddon(XSDGenerationAddon.class);
        if (xga != null)
        {
            xga.addMethod(pi,m_method);
        }
    }
}

```

The above code does the following:

- Some `td` and `div` elements are created. This is where the Silverlight object is placed in. The `div` receives a certain ID ("Cxxx") which will be referenced later. In addition, there is a JavaScript function `onButtonPressedxxx()` that will later be called by the Silverlight control.

- In the JavaScript initialization, the variable reference "C_xxx" is defined, which points to the div. The actual generation of the Silverlight object is then done by using the function `createMySilverlightControl`. Some parameters are transferred, for example, the name of the XAML file to render. The `createMySilverlightControl` function is part of the *createSilverlight.js* file and internally wraps the creation of the Silverlight control:

```
function createMySilverlightControl(pXaml,pDiv,pId,pWidth,pHeight)
{
    Sys.Silverlight.createObject(
        pXaml,
        pDiv,
        "SL_" + pId,
        {
            width:pWidth,
            height:pHeight,
            inplaceInstallPrompt:false,
            background:'#FFFFFF',
            isWindowless:'false',
            framerate:'24',
            version:'0.9'
        },
        {
            onError:null,
            onLoad:null
        },
        pId);
}
```

This may look complex. However, when you look at the Silverlight SDK documentation, you see that this is just a normal Silverlight implementation. The `Sys.Silverlight.createObject` function checks whether Silverlight is installed and creates a corresponding `object` tag into the surrounding `div` element that is passed as parameter.

- Finally you see that the `C_xxx` variable gets some attributes which take over definitions such as width and height to be used later on.

The control is rendered and Silverlight is invoked.

XAML File

Let us now have a look at the XAML file that is passed as parameter. The XAML file is the XML definition which defines the layout of what is rendered inside the Silverlight control:

```

<Canvas
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<Canvas Canvas.Top="0"
  Canvas.Left="0"
  Width="100"
  Height="21"
  Loaded="onLoadedBUTTON">
  <Canvas Canvas.Top="0"
    Canvas.Left="0"
    Width="100"
    Height="21"
    x:Name="l1l1l1"
    MouseLeftButtonDown="onMouseLeftButtonDownBUTTON">
    <Canvas.Resources>
      <Storyboard x:Name="button_flip">
        <DoubleAnimation
          Storyboard.TargetName="l1l1l1scale"
          Storyboard.TargetProperty="ScaleY"
          From="1.0" To="-1.0" Duration="0:0:0.400" AutoReverse="True"/>
        </Storyboard>
      </Canvas.Resources>
      <!-- Content Begin -->
      <Rectangle Canvas.Top="0"
        Canvas.Left="0"
        Width="100"
        Height="21"
        Stroke="#808080"
        StrokeThickness="1"
        RadiusX="10"
        RadiusY="10"
        Cursor="Hand">
        <Rectangle.Fill>
          <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <GradientStop Color="#E0E0E0" Offset="0.0" />
            <GradientStop Color="#A0A0A0" Offset="1.0" />
          </LinearGradientBrush>
        </Rectangle.Fill>
      </Rectangle>
      <TextBlock Text=""
        FontSize="12"
        Canvas.Top="1"
        Canvas.Left="10"
        Cursor="Hand"
        x:Name="text1"/>
      <TextBlock Text=""
        FontSize="12"
        Canvas.Top="22"
        Canvas.Left="10"
        Height="5"
        Cursor="Hand"

```

```

        x:Name="text2">
        <TextBlock.Foreground>
            <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                <GradientStop Color="#B0B0B0" Offset="0.0" />
                <GradientStop Color="#606060" Offset="1.0" />
            </LinearGradientBrush>
        </TextBlock.Foreground>
        <TextBlock.RenderTransform>
            <ScaleTransform ScaleX="1" ScaleY="-0.6"/>
        </TextBlock.RenderTransform>
    </TextBlock>
    <!-- Content End -->
    <Canvas.RenderTransform>
        <ScaleTransform x:Name="llllscale" ScaleX="1" ScaleY="1" CenterY="10"/>
    </Canvas.RenderTransform>
</Canvas>
</Canvas>
</Canvas>

```

If this is the first XAML definition that you see, it may look confusing. But in the end, it is quite simple what happens:

- There is a set of `Canvas` definitions, each one representing a drawing zone of its own.
- There is a rectangle with rounded edges (that is: the button) with a brush definition.
- There are two text block definitions holding a text which will later be passed by JavaScript. One text block is the “main text”, the other one is a “mirrored text” (the button’s text is mirrored in a light gray color).
- There is a `ScaleTransform` at the very end.
- And there is a `Storyboard` definition in the resources section, which is used to scale the transformation of the Y coordinates from 1 to -1 in 400ms, and back. This is what happens when the user chooses the button: the button flips vertically and goes back to its original position.

JavaScript

Finally, there is some JavaScript that plays a role: one canvas contains an event handler `MouseLeftButtonDown` which calls a method `onMouseLeftButtonDownBUTTON`. The implementation of this method is done in the file *creatSilverlight.js*. This is the file from which the control was created.

```

function onMouseLeftButtonDownBUTTON(sender,eventArgs)
{
    var control = sender.getHost();
    var params = control.initParams;
    // animation
    var flippig = control.content.findName("button_flip");
    flippig.begin();
    // invoke button function
    var methodName = "onButtonPressed"+params;

```



```

    var method = window[methodName];
    method();
}

```

The following happens:

- The flipping is started by referencing the corresponding ID "button_flip" from the XAML file and calling the corresponding function.
- The method `onButtonPressedxxx` is started inside the HTML pages. This is the one that is generated by the control handler. The method itself is calling a server-side function. It uses the JavaScript API that is available for Application Designer controls.

There is one thing left which was not mentioned yet: the setting of the button's text. This is done through the following JavaScript implementation:

```

function onLoadedBUTTON(sender,eventArgs)
{
    var control = sender.getHost();
    var params = control.initParams;
    var cc = window["C_"+params];
    var text1 = control.content.findName("text1");
    text1.text = cc.CASA_name;
    var text2 = control.content.findName("text2");
    text2.text = cc.CASA_name;
    var taw = text1.actualWidth;
    var centeredX = 50 - taw/2;
    text1["Canvas.Left"] = centeredX;
    text2["Canvas.Left"] = centeredX;
}

```

This is called when the button is loaded inside the Silverlight control. It picks the method name and passes it to the text blocks within the XAML definition.

SL_IMAGEOUT

The implementation of `SL_IMAGEOUT` is similar to that of `SL_BUTTON`.

Editor Extension File

This is the control's editor extension file:

```

<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

    <!--
    *****
    * TAGs
    *****
    -->

    <!-- SL_SLIMAGEOUT -->
    <tag name="sl_imageout">
      <attribute name="valueprop"/>
      <attribute name="width" datatype="width"/>
      <attribute name="height" datatype="height"/>
      <taginstance/>
      <protocolitem/>
    </tag>
    <tagsubnodeextension control="itr" newsubnode="sl_imageout"/>
    <tagsubnodeextension control="tr" newsubnode="sl_imageout"/>
    <taggroupsubnodeextension group="Silverlight" newsubnode="sl_imageout"/>

  </editor>
</controllibrary>

```

The control defines three attributes. `width` and `height` are used to set the image dimension. `valueprop` is used to bind the image's URL to an adapter property.

In the controls palette of the Layout Painter, the control will appear within the section **Silverlight**. It can be placed below ITR and TR row containers.

Control Handler

This is the tag handler implementation:

```

package com.softwareag.cis.gui.generate;

import org.xml.sax.AttributeList;

import com.softwareag.cis.gui.protocol.ProtocolItem;

public class SL_IMAGEOUTHandler implements ITagHandler
{
    String m_valueprop;
    String m_width = "300";
    String m_height= "300";

```

```

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

public void generateHTMLForStartTag(int id,
                                   String tagName,
                                   AttributeList attrlist,
                                   ITagHandler[] handlersAbove,
                                   StringBuffer sb,
                                   ProtocolItem protocolItem)
{
    readAttributes(attrlist);
    fillProtocol(protocolItem);
    generateXSD(protocolItem);
    // create HTML
    sb.append("<!-- SL_IMAGEOUT begin -->\n");
    sb.append("<td width='"+m_width+"'><div id='C"+id+"' ↵
style='width:"+m_width+"; height:"+m_height+"'></td>\n");
    sb.append("<script>\n");
    sb.append("var s_vv"+id+" = 'inittini';\n");
    sb.append("function romu"+id+"()\n");
    sb.append("{\n");
    sb.append("    var vv = C.getPropertyValue('"+m_valueprop+"');\n");
    sb.append("    if (vv == s_vv"+id+") return;\n");
    sb.append("    s_vv"+id+" = vv;\n");
    sb.append("    displayImageIMAGEOUT(C"+id+",vv);\n");
    sb.append("}\n");
    sb.append("</script>\n");
}

public void generateHTMLForEndTag(String tagName, StringBuffer sb)
{
    sb.append("<!-- SL_IMAGEOUT end -->\n");
}

public void generateJavaScriptForInit(int id,
                                     String tagName,
                                     StringBuffer sb)
{
    sb.append("if (firstusage) C_"+id+" = document.getElementById('C"+id+"');\n");
    sb.append("if (firstusage) C_"+id+".CASA_width='"+m_width+"';\n");
    sb.append("if (firstusage) C_"+id+".CASA_valueprop='"+m_valueprop+"';\n");
    sb.append("if (firstusage) C_"+id+".CASA_width='"+m_width+"';\n");
    sb.append("if (firstusage) C_"+id+".CASA_height='"+m_height+"';\n");
    sb.append("if (firstusage) ↵
createSilverlightControl('../HTMLBasedGUI/silverlight/SL_IMAGEOUT.xaml',C_"+id+", '"+id+"', '"+m_width+"', '"+m_height+"');\n");
    sb.append("if (firstusage) C.registerListener(romu"+id+");\n");
}

// -----
// private usage

```

```
// -----

/** */
private void readAttributes(AttributeList attrlist)
{
    for (int i=0; i<attrlist.getLength(); i++)
    {
        if (attrlist.getName(i).equals("valueprop")) m_valueprop = ↵
attrlist.getValue(i);
        if (attrlist.getName(i).equals("width")) m_width = attrlist.getValue(i);
        if (attrlist.getName(i).equals("height")) m_height = attrlist.getValue(i);
    }
}

/** */
private void fillProtocol(ProtocolItem pi)
{
    pi.addProperty(m_valueprop);
}

/** */
private void generateXSD(ProtocolItem pi)
{
    XSDGenerationAddon xga = ↵
(XSDGenerationAddon)pi.findGenerationAddon(XSDGenerationAddon.class);
    if (xga != null)
    {
        xga.addSimpleData(pi,m_valueprop);
    }
}
}
```

This is similar to `SL_BUTTON`. However, there is a `romuxxx` function generated and registered via `C.registerListener()`. This function is called every time a response is processed within the client. Inside the method, the data that was updated by the response is checked whether it contains a new image to be displayed in the control.

Now, let us have a look at the XAML implementation:

```
<Canvas
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Loaded="loadedIMAGEOUT">

    <Canvas.Resources>
        <Storyboard x:Name="thestoryboard1"
            Completed="completedIMAGEOUT">
            <DoubleAnimation
                Storyboard.TargetName="tr"
                Storyboard.TargetProperty="Angle"
```

```

        From="0.0" To="45" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="ScaleX"
        From="1" To="0" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="ScaleY"
        From="1" To="0" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="CenterY"
        From="0" To="500" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="CenterX"
        From="0" To="500" Duration="0:0:0.400" AutoReverse="False"/>
</Storyboard>
<Storyboard x:Name="thestoryboard2">
    <DoubleAnimation
        Storyboard.TargetName="tr"
        Storyboard.TargetProperty="Angle"
        From="45" To="0" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="ScaleX"
        From="0" To="1" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="ScaleY"
        From="0" To="1" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="CenterY"
        From="500" To="0" Duration="0:0:0.400" AutoReverse="False"/>
    <DoubleAnimation
        Storyboard.TargetName="ts"
        Storyboard.TargetProperty="CenterX"
        From="500" To="0" Duration="0:0:0.400" AutoReverse="False"/>
</Storyboard>
</Canvas.Resources>

<Rectangle Canvas.Top="0"
    Canvas.Left="0"
    Width="300"
    Height="300"
    Stroke="#606060"
    StrokeThickness="1"
    RadiusX="10"
    RadiusY="10">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">

```

```

        <GradientStop Color="#E0E0E0" Offset="0.0" />
        <GradientStop Color="#909090" Offset="1.0" />
    </LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>

<Canvas Canvas.Top="10"
    Canvas.Left="10"
    Width="280"
    Height="280">
    <Rectangle Canvas.Top="0"
        Canvas.Left="0"
        Width="280"
        Height="280"
        Cursor="Hand">
        <Rectangle.Fill>
            <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                <GradientStop Color="#808080" Offset="0.0" />
                <GradientStop Color="#000000" Offset="1.0" />
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>
    <Canvas Canvas.Top="0"
        Canvas.Left="0"
        Width="280"
        Height="280">
        <Canvas.Background>
            <ImageBrush x:Name="imagebrush" ↵
ImageSource="../../../HTMLBasedGUI/silverlight/SL_IMAGEOUT_empty.jpg" />
        </Canvas.Background>
        <Canvas.RenderTransform>
            <TransformGroup>
                <ScaleTransform x:Name="ts" ScaleX="1" ScaleY="1" CenterY="0"/>
                <RotateTransform x:Name="tr" Angle="0"/>
            </TransformGroup>
        </Canvas.RenderTransform>
    </Canvas>
</Canvas>
</Canvas>

```

There is a set of canvas definitions (see the screenshot at the very beginning of this document: the image is embedded in some kind of frame). The most important one is located at the bottom: it uses an "ImageBrush" in order to paint its content (this means: an image appears in the content).

There is a set of animations (two storyboards) which define that the image rolls out and in by doing a scale transformation and a rotate transformation simultaneously.

The following is the JavaScript code which does the binding from JavaScript to Silverlight:

```

function displayImageIMAGEOUT(cc,imageName)
{
    if (imageName == null || imageName == "")
    {
        return;
    }
    cc.CASA_imageName = imageName;
    if (cc.CASA_control == undefined) // not yet initialized
    {
        return;
    }
    var control = cc.CASA_control;
    // start the hide storyboard
    var tsb = control.content.findName("thestoryboard1");
    tsb.begin();
}

function loadedIMAGEOUT(sender,args)
{
    var control = sender.getHost();
    var params = control.initParams;
    var cc = window["C_"+params];
    cc.CASA_control = control;
    if (cc.CASA_imageName != null)
        displayImageIMAGEOUT(cc,cc.CASA_imageName);
}

function completedIMAGEOUT(sender,args)
{
    var control = sender.getHost();
    var params = control.initParams;
    var cc = window["C_"+params];
    var ib = control.content.findName("imagebrush");
    ib.imageSource = cc.CASA_imageName;
    var tsb = control.content.findName("thestoryboard2");
    tsb.begin();
}

```

When a new image needs to be displayed, the `displayImageIMAGEOUT` method is called. This is the one which is referenced by the `romuxxx` method which itself is called when a response is processed on the client side.

The `displayImageIMAGEOUT` method triggers the animation `thestoryboard1`. This animation takes the current image out (that is: it rotates and shrinks it). At the end of the animation, an event is raised which then calls the function `completedIMAGEOUT`. In the `completedIMAGEOUT` method, the new image is defined in the brush, and the animation is started to roll the new image in.

78

Integrating Application Designer Controls in HTML Pages

■ Example	376
■ Details on the Implementation	378
■ Invoking the Page in the Browser	379
■ PGHEAD Properties	380
■ PGCONTAINER Properties	381

Application Designer provides an “outside-in approach” which allows you to integrate Application Designer controls and functionality in standard HTML pages using a standard HTML editor or text editor.




Important: The HTML document containing your Application Designer controls must be XHTML-formatted. As a rule, XHTML format has to be switched on manually.

Example

The following is a standard HTML page that contains a registration form for a technical discussion forum. This registration form contains Application Designer controls.

PRODUCTS	SERVICES	COMMUNITY	ABOUT US
----------	----------	-----------	----------

| TECHNOLOGY FORUM | CUSTOMER FEEDBACK | CAREERS | USER GROUPS |
HOME |


Technology Discussion Forum Registration

Registration Information

E-Mail Address : *

Display Name : *

Password : *

Confirm Password : *

Profile Information

ICQ Number :

AIM Number :

Website :

Location :

Occupation :

Preferences

Newsletter : ☒ Like to receive a regular Newsletters

Always notify me of repies : ☐ Yes ☒ No

Always attach my signature : ☐ Yes ☒ No

Always allow HTML : ☒ Yes ☐ No

Board Language :

Board Style :




Time Zone :

Security Control Panel

It is important you think before filling this in. You will not be able to change these at will. You will need an admins approval to change them, for security purposes.

Security Question : *

Security Answer : *

COPYRIGHT©2007
SOFTWARE AG (INDIA) PVT. LTD
ALL RIGHTS RESERVED


Search:


The layout of the above page is defined in the following way:

```
<html>
<head>
<pghead stylesheetfile="../cis/styles/CIS_DEFAULT.css"></pghead>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>TechForum Registration </title>
</head>
<body>
<table>...</table>

    // standard HTML coding here

<pgcontainer model="RegistrationAdapter">
    // normal Application Designer XML coding
    <pagebody>
        ...
    </pagebody>
    <statusbar>
    </statusbar>
</pgcontainer>

    // standard HTML coding here

</body>
</html>
```

Details on the Implementation

To include Application Designer controls in an HTML page, you simply use an HTML editor or text editor to write the HTML code and then you include the following HTML elements in this code:

■ PGHEAD

This element is placed in the HTML header.

In the above example, the following code is used:

```
<pghead stylesheetfile="../cis/styles/CIS_DEFAULT.css">
```

The `stylesheetfile` attribute references the style sheet file that is to be used for the Application Designer controls. In this example, it references the standard `CIS_DEFAULT.css` file.

Make sure to specify a valid file reference. Have a close look at this reference, especially when your web application uses one or more subdirectories (for example, when your page is placed in `<web-application>/sub1/sub2/mypage.html`).

■ PGCONTAINER

This element is placed in the HTML body. You put the Application Designer controls into this element. The code is the same as for a standard Application Designer page.

Make sure you have valid Application Designer XML in this container element. We strongly recommend that you use Application Designer's XSD file *editor.xsd* for XML validation. For information on how to get an up-to-date *editor.xsd*, see *XML Schema (XSD)* in the *Development Workplace* documentation.

In the above example, the following code is used:

```
<pgcontainer model="RegistrationAdapter">
```

The `model` attribute references the associated adapter in which you use standard Application Designer classes:

```
// This class is a generated one.
...
public class RegistrationAdapter
extends Adapter
{
    /** initialisation - called when creating this instance*/
    public void init()
    {
        ...
    }
}
```

Invoking the Page in the Browser

In the browser, an HTML page containing Application Designer controls is addressed the same way as a normal Application Designer page. For example:

<http://localhost:51000/cis/servlet/StartCISPage?PAGEURL=/cismyproject/mypage.html>

PGHEAD Properties

Basic			
stylesheetfile	URL of a style sheet file used for control rendering. i.e. '../cis/styles/CIS_DEFAULT.css'.	Obligatory	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	<p>URL of an additional style sheet file.</p> <p>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:</p> <p>(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).</p> <p>(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.</p> <p>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".</p>	Optional	css
openajaxsupport	Adds registration code into the page that registers globally used objects / events etc. to the Open AJAX Hub in order to potentially synchronize the co-existence of different toolsets within one page. Only used when being familiar with OpenAJAX aspects.	Optional	true false

PGCONTAINER Properties

Basic			
model	<p>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.</p> <p>Example: if you have a class DemoAdapter inside the package com.xyz.demo, the MODEL value is: com.xyz.demo.DemoAdapter.</p> <p>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.</p> <p>You may use the class "DummyAdapter" for testing your layout - before specifying your "real" class.</p>	Optional	
translationreference	<p>This is the "translation reference" that is passed to the multi language management.</p> <p>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.</p> <p>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.</p> <p>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.</p>	Optional	

	<p>If you on the other hand define one translation reference for multiple pages then you can share text-ids throughout the various pages.</p> <p>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".</p>		
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	<p>Each CIS page can be opened as a popup dialog. This property defines the pixel height preferred for the page.</p> <p>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.</p>	Optional	100px 200px 300px 400px
popupfeatures	<p>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.</p> <p>center:yes no</p> <p>edge:sunken raised</p> <p>resizable:yes no</p> <p>scroll:yes no</p> <p>status:yes no (to display or hide a status bar)</p> <p>An example string looks as follows: "dialogLeft:100px"</p> <p>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</p>	Optional	dialogLeft: 200px dialogTop: 100px edge: sunken resizable: yes status: no

	<p>from the left top corner of the current window. - Use "dialogTop: SCRY(100)px" in the same way for vertical positioning.</p> <p>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.</p>		
imagestopreload	<p>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.</p> <p>The URL of the images must be relative to your generated HTML page.</p> <p>Example: if your page has a tree with certain node images then you may define: "images/nodeopened.gif" images/nodeclosed.gif; images/nodeendnode.gif".</p>	Optional	
occupiedimage	<p>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.</p> <p>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.</p>	Optional	
occupiedpixelheight	<p>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.</p>	Optional	
occupiedpixelwidth	<p>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.</p>	Optional	
helpid	<p>This is the id that is passed into the help management for the page.</p> <p>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.</p>	Optional	

visiblevalueifundefined	<p>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.</p> <p>This property defines how these controls behave if there is no implementation available for the property.</p> <p>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.</p>	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	
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. texgrid, tree,...) handled the click so far.	Optional	
hotkeys	<p>Comma 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 comma</p> <p>Example:</p> <p>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.</p> <p>Use the popup help within the Layout Painter to input hot keys.</p>	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
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	
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	
adapterlisteners	Semicolon separated list of classes which connect to the server side adapter processing as adapter listeners (each one supporting the interface IAdapterListener).	Optional	

79 Automated Testing

An Application Designer application usually consists of layout pages, corresponding Java or Natural adapters and additional server-side Java or Natural implementation. For testing the server-side implementations that are not related to a user interface, good test coverage can be achieved, for example, with JUnit tests. To achieve reasonable test coverage for the whole application, however, automated tests for the parts which relate to the user interface are also required.

For testing web applications, tools such as Selenium (see <http://docs.seleniumhq.org/>) are available. In automated tests, the test cases have to locate different elements in an HTML page. The tests usually use identifiers in order to locate the elements for testing. When an Application Designer layout page is published, the identifiers are automatically generated in the corresponding HTML page using continuous numbering. However, identifiers which are based on continuous numbers are not suitable for automated tests. Even a small change such as rearranging fields or adding an additional field will change the corresponding number and break the test. Therefore, it is more suitable to use stable identifiers in the test cases. With stable identifiers, the tests will still run correctly when small changes have been applied to the layout.

With Application Designer, you can add stable identifiers (test tool IDs) to the controls. In the Layout Painter, the **Miscellaneous** tab of the properties area contains the `testtoolid` property. The value for this property can be chosen freely. It is not used within the Application Designer framework.

The value of the `testtoolid` is generated into the corresponding generated HTML page and the element can thus be located by the test program in a stable way.

The following shows an extract from the generated HTML. You can see the difference between the unstable `id` value and the stable `testtoolid` value. When the field is rearranged on the page, the `id` value will change. The `testtoolid` value, however, will stay the same.

```
<input id="F_28" name="CC" class='FIELDInputEdit'  
testtoolid='mytestid' type="text" size='15' style=" ">
```

With the tool Selenium, for instance, you can locate the FIELD control using a corresponding XPATH expression which contains the `testtoolid` value. This XPATH will still be valid even if you add fields to the layout.