software AG

Application Designer

Working with Pages

ADABAS & NATURAL

Table of Contents

Preface

This documentation deals with more complex applications in which you have sequences of pages. The information is subdivided into the following parts:

Working with Page Navigation	Describes how to develop a sequence for page navigation.
Embedding Pages into Pages	Describes how to develop a master page and embed other pages within it.
Multi Frame Pages	Describes how to generate a HTML frameset page.
Embedding Pages into a Workplace	Describes how to integrate pages into portal or workplace environments.

1 About this Documentation

Document Conventions

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

Online Information and Support

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   Working with Page Navigation

In more complex applications, you often have to cope with situations in which you have to go through a sequence of pages. For example, for entering a purchase order, you have to specify first some header information (like customer, address, etc.), go to a list of items you want to order, open detail information page(s) on a selected item, etc.

The navigation can be quite complex on its own - there are several frameworks available which deal with this topic.

What Application Designer offers is a way to navigate between different pages. How you find out when and where to navigate to (server-side business logic) - is not of interest for Application Designer. As soon as you know where to go to, tell Application Designer your decision.

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

Page Navigation
Session Management
Opening Modal Pop-up Dialogs
URL to Choose when Navigating
Value Help Pop-up Dialogs
Standard Pop-up Dialogs
Page-based Pop-up Dialogs

2   **Page Navigation**

Page navigation is triggered by the adapter class. Typically, it is a method call which is triggered by a button click. The `Adapter` class from which you derive your adapter class, offers some methods which make page navigation very simple.

The First Navigation

The most simple way of navigating can be seen in the following code example:

```
public void showNextPage()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Please first input all fields");
        return;
    }
    // open new page
    this.switchToPage("pageName.html");
}
```

In this method, first there is a check whether navigation is "appropriate" in the current situation. If not, an error message is shown in the status bar. Otherwise, navigation is done by the inherited method `switchToPage(pageName)`.

Preparing the Adapter before Navigating

In our example, the next screen is - as usual - linked with a specific adapter class. An instance of this adapter class is generated by the session management.

If you want to prepare the adapter of the next screen in a certain way, you proceed as follows.

Before navigating to the next page, you can ask for the adapter which is linked to the next page:

```
public void showNextPage()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Please first input all fields");
        return;
    }
    // prepare adapter object which corresponds to next page
    XYZModel m = (XYZModel)this.findAdapter(XYZModel.class);
```

```
    m.setParam1(...);
    m.setParam2(...);
    ...
    ...
    // open new page
    this.switchToPage("pageName.html");
}
```

The method `findAdapter` returns the adapter object which is assigned to the next page. Therefore, you are able to prepare the adapter by setting any information you want to show in the next screen.

Including the Adapter while Navigating

The following example shows how to increase performance for page navigation using the method `includeAdapterInResponse`. It is typically used with:

- `switchToPage();`

- `openPopup();`

- `openCISPageInTarget()`

Example:

```
public void showNextPage()
{
    // open new page
    switchToPage("pageName.html");
    // initialize the corresponding Adapter of pageName.html
    includeAdapterInResponse("pageName.html", true);
}
```

The method `includeAdapterInResponse` includes the adapter of the second page (page B) into the response processing of the first page (page A). The adapter is processed in the same way as it is processed when being called by an explicit HTTP request coming from the browser. This is an effective mechanism for reducing the number of roundtrips between the browser and the server (it reduces the number of roundtrips from two to one). This is illustrated by the following diagram:

Page (Browser)   PageAAdapter   PageBAdapter

Button click

Invoke method

switchToPage("b.html")

includeAdapterInResponse("b.html")

init()

① Optimized roundtrip behavior: more performance by reducing the number of roundtrips.

② Standard behavior: loading of PageBAdapter in a second roundtrip.
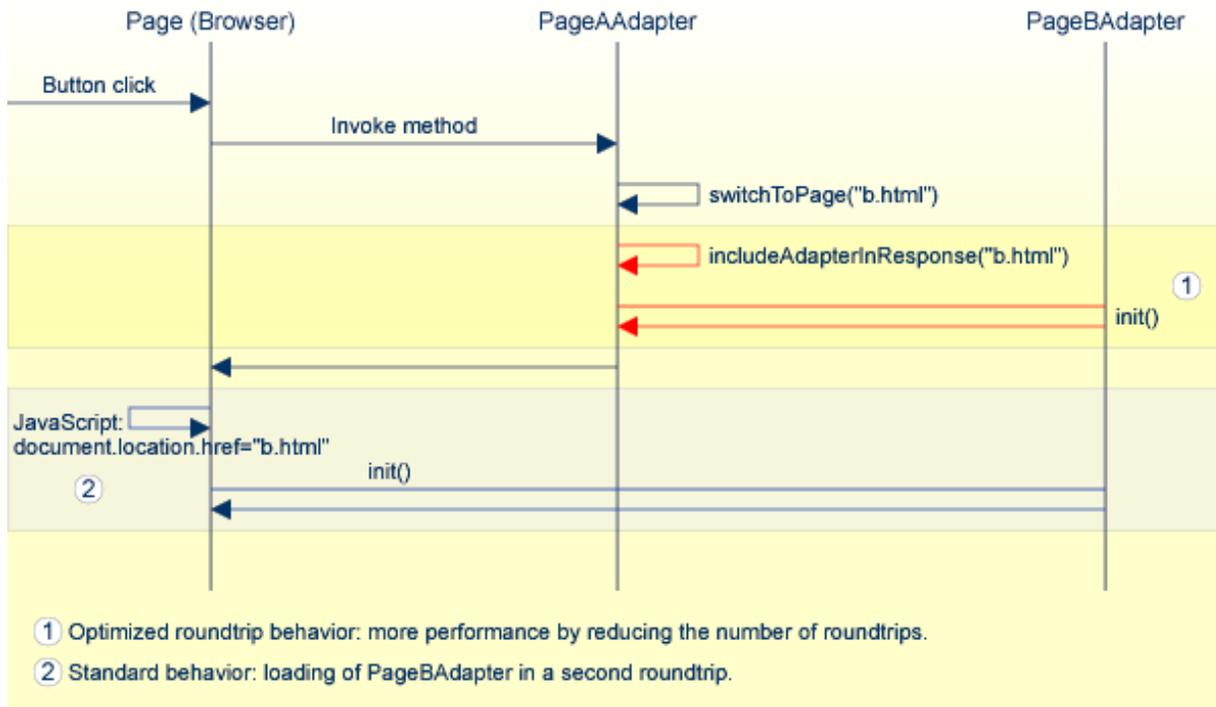
Note: In a local area network (LAN) environment, the gain in performance will not be significant. However, in a slow wide area network (WAN) environment, the performance will be improved significantly.

3   **Session Management**

You might ask: who controls the life cycle of the adapter classes? If I navigate from page "A" to page "B" and go back to page "A": do I come back to the adapter object I was already using, or do I get a new adapter instance?

Session, Subsession, Adapter

The management of the adapters inside the server is done by the session management of Application Designer. Typically you do not have to take care of it - it is done automatically in front of your adapters.

Every browser instance connected to Application Designer creates a session and is assigned to it at the server side. If you start another browser instance, a second session is created internally which is completely decoupled from all other sessions. And so on.

A session is divided into subsessions. A subsession is a logical separation of independent activities which run parallel within the context of one session. Example: in the workplace, you can run various applications in parallel. You can switch from one application to the other. Each running application is represented by an instance of a subsession at the server side. The subsessions are also completely isolated from each other.

Within a subsession, the adapter instances are held. The basic rules for managing these instances inside one subsession are:

- For each adapter class one instance is kept. This means: if a page requests an adapter, it is first determined whether this adapter instance is already created within the subsession. If yes, the existing instance is used, otherwise a new adapter instance is created and registered.

- The adapter instance is held for the whole life cycle of the subsession - as long as not explicitly removed by the adapter logic.

- All variant and page navigation is done inside a subsession as described in this section.

Session Management in Application Designer
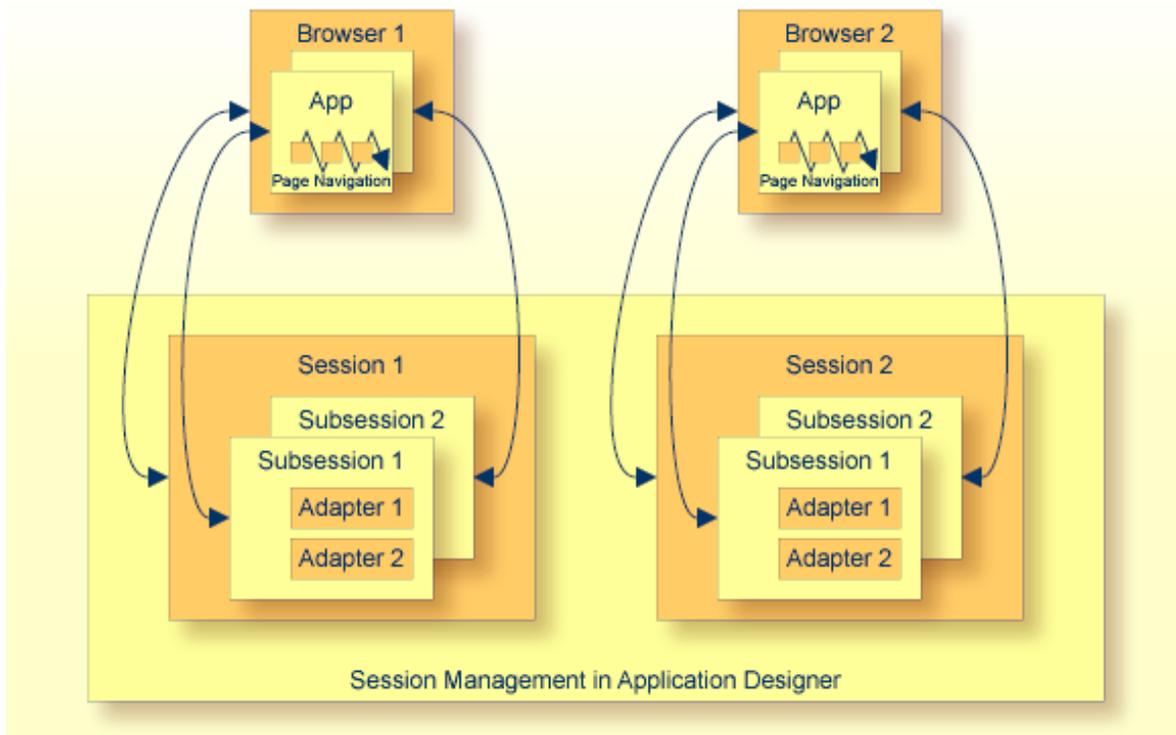
Page navigation within the browser is a navigation between adapter instances of the same subsession.

Garbage Collection

The final garbage collection of adapter instances is done by removing a subsession - if not explicitly controlled in a different way by the adapter logic. The adapter class offers the method `endProcess()` which removes the subsession you are just working with:

```
public void exit()
{
    // check if you really want to exit
    if ( ... )
    {
        ...
        this.outputMessage("E","Cannot exit due to...");
        return;
    }
    // exit
    this.endProcess();
}
```

Whenever a user logs off, the session - including all subsessions and its assigned adapter instances - is removed from the session management and released for garbage collection.

4   Opening Modal Pop-up Dialogs

Pop-up dialogs are just normal Application Designer pages (except for a small difference) which are opened in modal pop-up mode. The pop-up management does not start a new browser instance - everything is done in the same instance in which you are working.

Invoking a pop-up dialog follows the same rules as navigating between pages. The Java source of the adapter looks as follows:

```
public void showPopup()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Opening pop-up is not possible...");
        return;
    }
    // open new page
    this.openPopup("pageName.html");
}
```

The adapter - which is used as a server-side counterpart of the pop-up dialog - is managed like navigating between pages. Therefore, you can access the adapter before opening the pop-up dialog and prepare some content:

```
public void showPopup()
{
    // check if navigation is possible
    if (... any check is wrong...)
    {
        ...
        this.outputMessage("E","Opening pop-up is not possible...");
        return;
    }
    // prepare adapter object which corresponds to next page
    XYZModel m = (XYZModel)this.findAdapter(XYZModel.class);
    m.setParam1(...);
    m.setParam2(...);
    ...
    ...
    // open new page
    this.openPopup("pageName.html");
}
```

Special Pop-up Dialog Parameters within the XML Layout Definition

Any Application Designer page can be opened as a pop-up dialog. Inside the PAGE tag of the page, you can define how to open the pop-up dialog. There are a couple of properties which can be used for this purpose:

- `popupwidth`

- `popupheight`

- `popupfeature`

For further information, see the PAGE property definition in *Typical Page Layout* which is part of the *Layout Elements* documentation.

Passing Pop-up Dialog Parameters before Opening a Pop-up

The pop-up parameters (width, height, features) can also be passed before calling a pop-up. The `Adapter` class offers corresponding interfaces. The following code shows how to open a pop-up with a certain title and with a certain size and position:

```
/** */
public void onOpenPopup()
{
    setPopupFeatures(100, // x-position
                     100, // y-position
                     300, // width
                     200, // height
                    ""   // additional features as string (see PAGE-POPUPFEATURES ↩
docu)
                    );
    setPopupTitle("This is the title of the pop-up");
    openPopup("25_PositionedPopup1.html");
}
```

The parameters you pass override the parameters that may be defined in the pop-up page's layout definition.

Closing a Pop-up Dialog

A pop-up dialog can be closed by its corresponding adapter by the `closePage()` method which is inherited from the `Adapter` class:

```
/** This method is bound to the exit button of the pop-up page. */
public void exitPopup()
{
    // check if can be closed
    ...
    ...
    // close pop-up
    this.closePage();
}
```

In addition, a user can always close a dialog by pressing ALT+F4 or by choosing the close icon at the top right corner of the window title. The adapter - both adapters, the pop-up adapter and the adapter of the page from which the pop-up dialog was called - are not informed about this action and so it always has to be taken into consideration that a pop-up dialog might be closed by the user.

Changing the Size within an Opened Pop-up

Sometimes you need to resize the pop-up in which the user is currently working. For example, you want to show additional information and therefore have to increase the height of the pop-up.

The following code is inside the adapter object that belongs to the opened pop-up page:

```
public void onXxxxxx()
{
    findFunctionsLivingPopup().setPopupSize(m_newWidth,m_newHeight);
}
```

5 URL to Choose when Navigating

By the `switchToPage(...)` and the `openPopup(...)` methods, a URL is passed as a string parameter to Application Designer for navigation. How can the URL be defined?

You can use relative links as long as the page to which you navigate is in the same directory as the page from which you navigate. This is especially important when navigating between pages which belong to the same application project. See also *Application Project Management* in the *Special Development Topics*.

If you want to navigate outside your project, you have to specify a link starting with the document root of your HTTP server. For each application project, a new context path is set up with the name of the project. Navigating from one project's file to another can be done by specifying the full URL like /*<project>*/*<projectfile.html>*.

Pages created without the project management (such as the Hello World example) are accessible by the default context /*HTMLBasedGUI*/.

6   Value Help Pop-up Dialogs

In case you want to provide help in form of a pop-up dialog - based on a certain field - Application Designer offers a technique for implementing a value selection help:

- The FIELD control offers the property `popupmethod`. With this property, a method of the adapter class is called whenever the user requests a value help inside the field - by pressing F4 or F7 in the field or by double-clicking on the field.
- A pop-up dialog opens displaying possible data selections.

The value help pop-up dialogs are just normal pop-up dialogs which just have a dedicated purpose.

Standard Method openIdValueHelp

Inherited from the `Adapter` class, there is a very simple way to provide a value help pop-up dialog. The method `openIdValueHelp` is implemented in a generic way and can be used as follows:

- In your adapter, implement a method `findValidValuesForXxx()`. Replace "Xxx" with the name of the property field.
- This method must return an array of `com.softwareag.cis.server.util.ValidValueLine` objects. This array contains pairs of IDs and values which are valid data options for the `Xxx` property.
- When requesting a value help for the corresponding field, a pop-up dialog displays the `ValidValueLine` objects which are passed back from your method. If the user selects an item in the pop-up dialog, the value is placed in the `setXXX` method of the property.

The following Java source shows an example:

```
// property >department<
   String m_department;
   public String getDepartment() { return m_department; }
   public void setDepartment(String value) { m_department = value; }

   public ValidValueLine[] findValidValuesForDepartment()
   {
       Vector v = new Vector();
       v.addElement(new ValidValueLine("EXEC","Executive Board"));
       v.addElement(new ValidValueLine("PRO1","Production Line1"));
       v.addElement(new ValidValueLine("PRO2","Production Line2"));
       v.addElement(new ValidValueLine("SALE","Sales"));
       ValidValueLine[] result = new ValidValueLine[v.size()];
       v.copyInto(result);
       return result;
   }
```

The XML layout looks as follows:

```
<rowarea name="Field with Value Help">
    <itr>
        <label name="Department" width="100">
        </label>
        <field valueprop="department" width="150" popupmethod="openIdValueHelp">
        </field>
    </itr>
</rowarea>
```

The result is a field which automatically opens a pop-up dialog when the user presses F4 or F7, or double-clicks on the field.
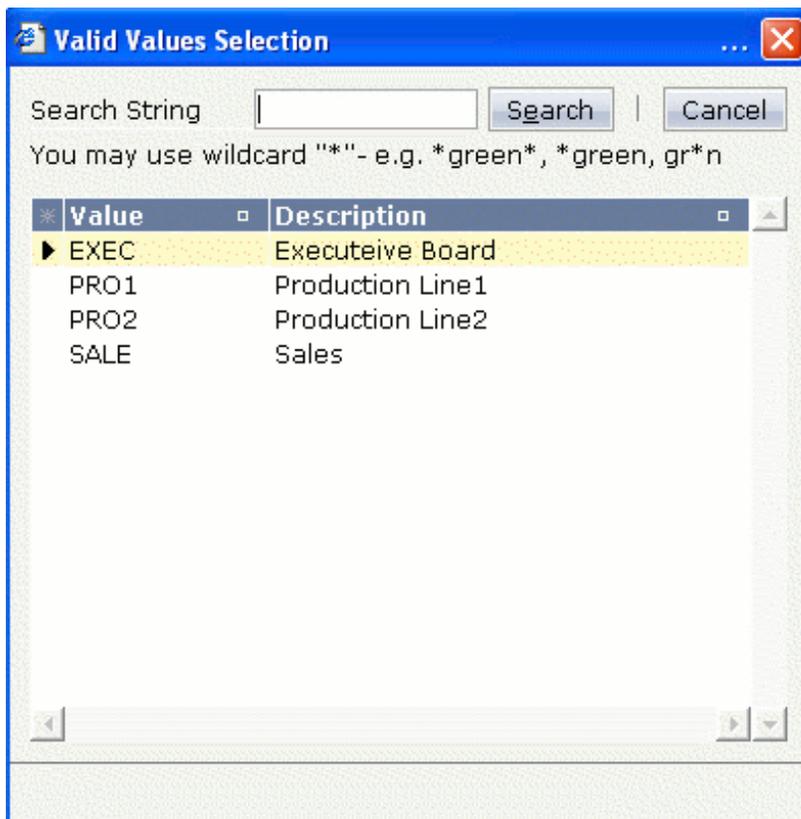
An additional feature available: instead of displaying pairs of ID and name, the dialog can display a list of IDs only. There is a constructor of the ValidValueLine class with only passing an ID to it:

```
public ValidValueLine[] findValidValuesForDepartment_02()
    {
        Vector v = new Vector();
        v.addElement(new ValidValueLine("EXEC"));
        v.addElement(new ValidValueLine("PRO1"));
        v.addElement(new ValidValueLine("PRO2"));
        v.addElement(new ValidValueLine("SALE"));
        ValidValueLine[] result = new ValidValueLine[v.size()];
        v.copyInto(result);
        return result;
    }
```

Now the pop-up dialog contains only a column containing the IDs:
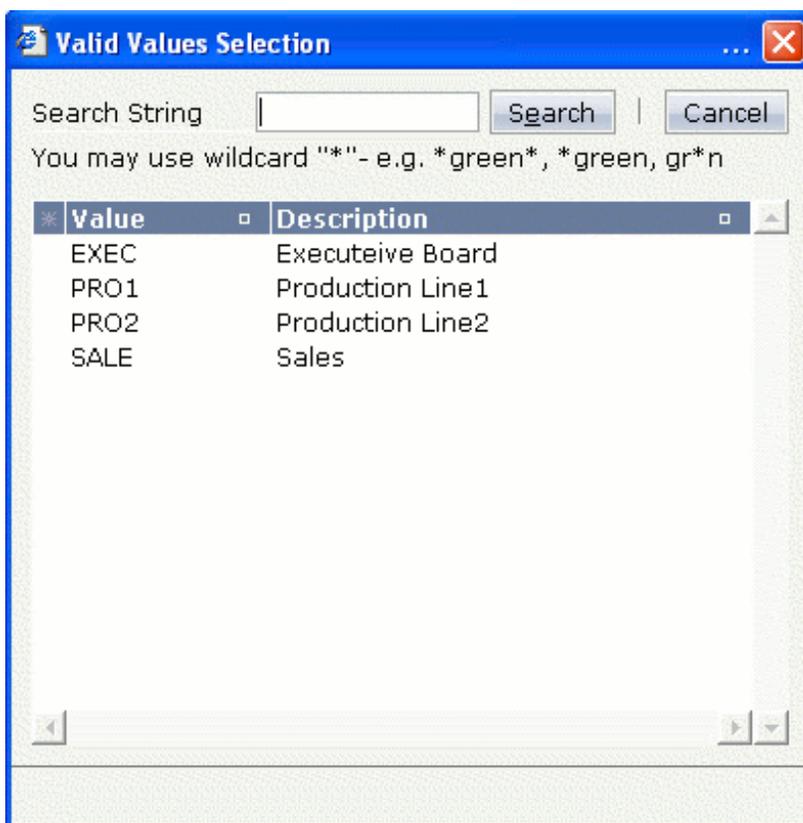
Standard Method openIdValueCombo

See the description of the FIELD control for information on how to implement a valid value help with the `openIdValueCombo` method. This method does not open a pop-up but it open a combo-like selection.

The interface on the server side is exactly the same as for `openIdValueHelp` - just the rendering result is different:
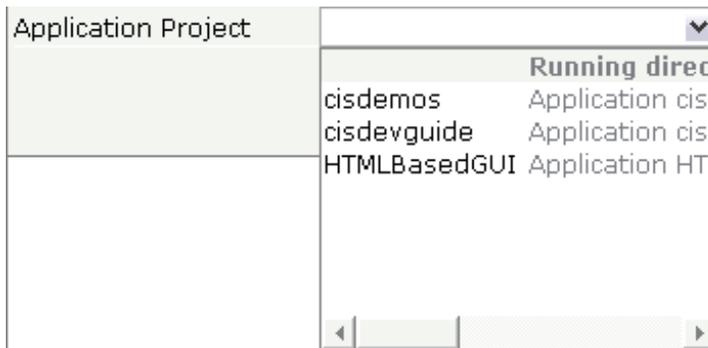
Further information is provided in the description of the FIELD control.

7 Standard Pop-up Dialogs

There are standard pop-up dialogs available for general usage which you do not have to code yourself.

OK Pop-up

The OK pop-up is used for displaying a text with an **OK** button.

The following is an example of an OK pop-up:
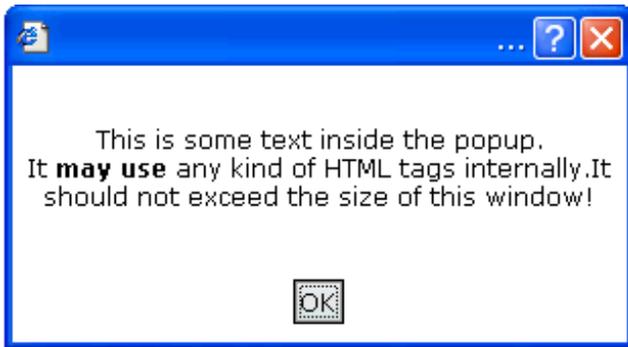
The code of the adapter is:

```
public void showOKPopup()
{
    PopupOKModel pok = (PopupOKModel)findAdapter(PopupOKModel.class);
    pok.init("This is some text inside the pop-up.<br>"+
            "It <b>may use</b> any kind of HTML tags internally." +
            "It should not exceed the size of this window!");
    this.openPopup("/HTMLBasedGUI/popupok.html");
}
```

Yes/No Pop-up

The Yes/No pop-up is used for asking the user a question. Depending on user's decision, activities are started inside the adapter.

The following is an example of a Yes/No pop-up:
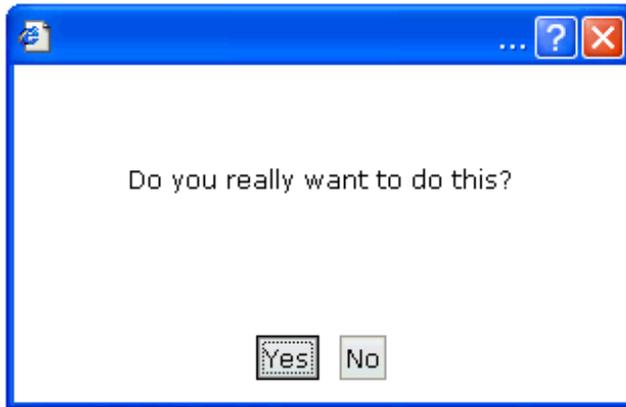
The code of the adapter is:

```
public class YESCommand implements com.softwareag.cis.server.util.ICommand
{
    public void execute()
    { outputMessage("S","Yes command was called"); }
}
public class NOCommand implements com.softwareag.cis.server.util.ICommand
{
    public void execute()
    { outputMessage("S","No command was called"); }
}
public void showYESNOPopup()
{
    PopupYesNoModel pyn = (PopupYesNoModel)findAdapter(PopupYesNoModel.class);
    pyn.init("Do you really want to do this?",
             new YESCommand(),
             new NOCommand());
    this.openPopup("/HTMLBasedGUI/popupyesno.html");
}
```

The pop-up dialog is initialised by passing the question and two "reaction objects" to it. One "reaction object" is called when choosing the **Yes** button, the other is called when choosing the **No** button.

The "reaction objects" have to implement the interface com.softwareag.cis.server.util.Icommand which just needs a simple execute() method. In our example, the "reaction objects" are implemented as inner classes of the adapter class.

Log Pop-up

The Log pop-up is used for displaying a log text.

The following is an example of a Log pop-up:
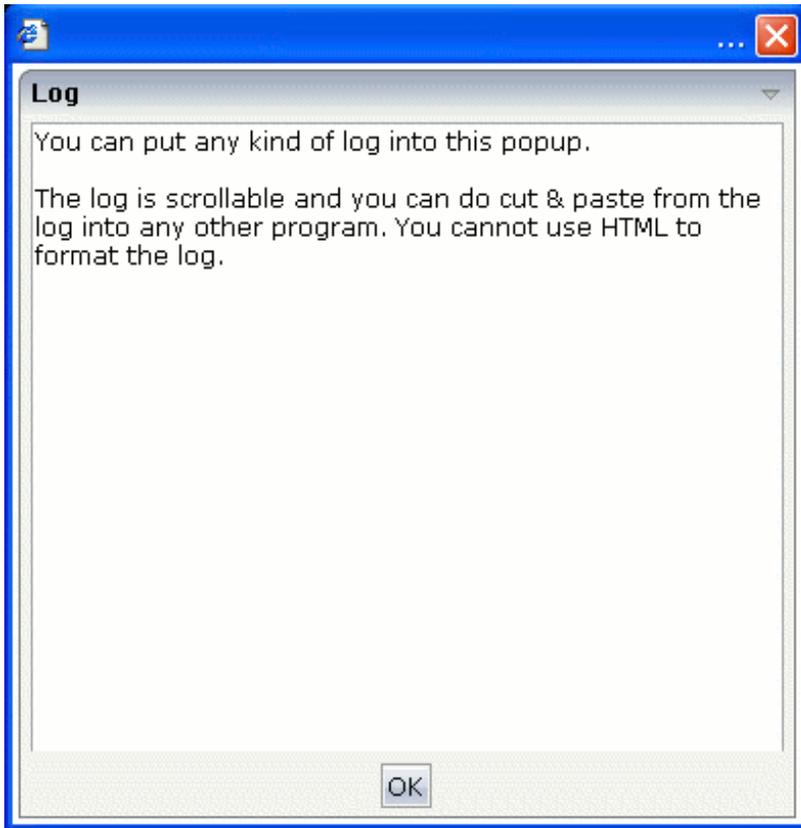
The code inside the adapter is:

```
public void showLOGPopup()
{
    PopupLogModel plm = (PopupLogModel)findAdapter(PopupLogModel.class);
    plm.init("You can put any kind of log into this pop-up.\n\n"+
            "The log is scrollable and you can do cut & paste from "+
            "the log into any other program. You cannot use HTML "+
            "to format the log.");
    this.openPopup("/HTMLBasedGUI/popuplog.html");
}
```

Example: Asking Whether the User Really Wants to Quit

This is a typical example: the user works on a page of your application for a while and then choose the close icon in the right top corner of the page. Check whether the user has changed something and ask using a pop-up dialog if the user really wants to close the page.

The following Java source shows an implementation in the adapter class:

```
/** */
public void endProcess()
{
    if (m_changed == true)
    {
        PopupYesNoModel pyn = (PopupYesNoModel)findAdapter(PopupYesNoModel.class);
        pyn.init("You modified some data. Do you really want to exit?",
                new ICommand() { public void execute() { executeEndProcess(); }},
                null);
        this.openPopup("/HTMLBasedGUI/popupyesno.html");
    }
    else
    {
        executeEndProcess();
    }
}

/** */
public void executeEndProcess()
{
    super.endProcess();
}
```

The `endProcess()` method is called by the closing function of the page. It is provided by the `Adapter` class from which the adapter is inherited. The `endProcess()` method does already everything which is required for removing the subsession.

Overwrite the `endProcess()` method and embed the code which opens a Yes/No pop-up to ask whether the user really wants to quit the application. The original closing function is shifted to the method `executeEndProcess()`. The Yes/No pop-up got for the "Yes" method an inner class pointing to the `executeEndProcess()` method. The "No" method is null and means that nothing should be done.

8 Page-based Pop-up Dialogs

Page-based pop-up dialogs look and behave different from the **standard pop-up dialogs**. The content of a page-based pop-up dialog is not opened within a modal browser window. This has the following advantages:

- Page-based pop-up dialogs are faster than the standard pop-up dialogs.

- There are no browser window drawbacks. A page-based pop-up dialog does not have a close icon; the user must always choose a command button before the page-based pop-up dialog is closed. There is no status bar in which an URL can be shown.

- Page-based pop-up dialogs are not affected by pop-up blockers.

- Page-based pop-up dialogs are page-modal only. This means that they do not block the whole browser window. Other pages can be opened in a subpage or in an other frame.
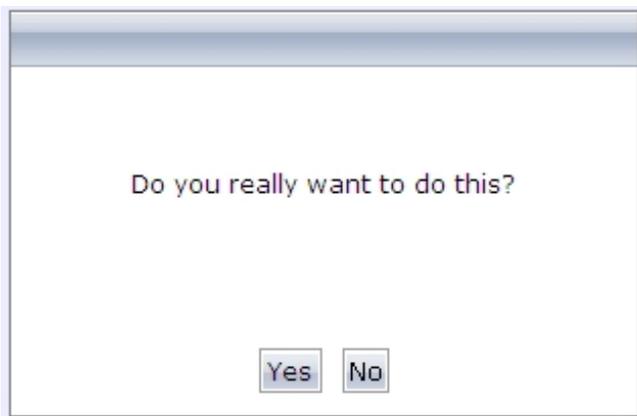
To open standard pop-up dialogs and page-based pop-up dialogs, you have to use the following API calls:

- Open a standard pop-up dialog in a browser window:

```
openPopup(page)
```

■ Open a page-based pop-up dialog:

```
openPagePopup(page)
```

II Embedding Pages into Pages

In Application Designer, there is the possibility to embed pages into other pages. Or vice versa: you can create pages which consist of other embedded pages. In this case, the (master) page and the embedding page operate independently from each other by having a channel to cooperate.

Use this technique to build pages for different scenarios. Example: embed a page showing an order detail into another page displaying a list of all orders. When you select an order from the list, it will be displayed in detail in the "inner" page.

The technology described in this part is very nice for modularising complex or large screens. It is not appropriate to use this technology for very fine modularisation, e.g. for just a couple of fields. (For more information on how to deal with "fine modularisation", see the *Custom Controls* documentation.)

There are two controls which support embedding of pages into pages:

- The SUBCISPAGE2 control represents a rectangular area inside a page in which another Application Designer page can be included.
- The ROWTABSUBPAGES control is a tab selection control where a dynamic set of pages can be arranged.

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

SUBCISPAGE2 Control

ROWTABSUBPAGES Control

Remark on Modularisation

9 SUBCISPAGE2 Control

The SUBCISPAGE2 control allows you to place one page into another page. You may already have read the section describing the SUBPAGE control which allows to place any HTML page into an Application Designer page. The differences between the SUBCISPAGE2 and the SUBPAGE tag are:

- With SUBCISPAGE2, you embed Application Designer pages, not normal HTML pages.
- Application Designer pages are normally started using a servlet "StartCISPage" which creates an embedding frame in which the Application Designer page is placed. The SUBCISPAGE2 control automatically creates this frame, you do not have to take care of this.
- There is a defined communication channel allowing the "outside page" to interact with the "embedded page", and vice versa.
- The embedded page is automatically linked to the Application Designer session management. It runs in the same session - and typically also in the same subsession as the embedding page.

Simple Example

The following example shows the input of an article number and its article detail data:

The detail data page is embedded into the whole (outer) page. The XML code of the outer page is:

```
<page model="OuterPageAdapter" pagename="Demo.html">
    <titlebar name="Page with embedded page">
    </titlebar>
    <header>
    </header>
    <pagebody takefullheight="true">
        <rowarea name="Page Name Input">
            <itr>
                <label name="Article" width="100">
                </label>
                <field valueprop="article" length="20">
                </field>
            </itr>
            <vdist height="5">
            </vdist>
            <itr>
```

```
                    <hdist width="100">
                    </hdist>
                    <button name="Show Details" method="showDetails">
                    </button>
            </itr>
            <vdist height="5">
            </vdist>
            <rowtable0>
                <itr width="100%">
                        <subcispage2 subcispageprop="innerPage" width="100%" ↵
height="350" borderwidth="1">
                        </subcispage2>
                </itr>
            </rowtable0>
            <vdist height="5">
            </vdist>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The SUBCISPAGE2 control references a property `innerPage` which is provided by the adapter class of the page. The height can be specified depending on the whole page's height or can be fixed.

The corresponding adapter source is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.SUBCISPAGEInfo;

public class SubCisPage2Adapter
 extends Adapter
{
    // property >innerPage<
    SUBCISPAGEInfo m_innerPage = new SUBCISPAGEInfo();
    public SUBCISPAGEInfo getInnerPage() { return m_innerPage; }

    // property >article<
    String m_article;
    public void setArticle(String value) { m_article = value; }
    public String getArticle() { return m_article; }

    /** */
    public void init()
    {
        m_innerPage.showPage("ArticlePage.html");
    }
```

```
    /** */
    public void showDetails()
    {
        // fetch adapter of inner page
        ArticlePageAdapter ipa = (ArticlePageAdapter) ↵
findAdapter(ArticlePageAdapter.class);
        ipa.init(m_article);

        // trigger a refresh of the innerpage
        m_innerPage.refreshContentOfCurrentPage();
    }
}
```

The property `innerPage` is of type `com.softwareag.cis.server.util.SUBCISPAGEInfo`. With method `SUBCISPAGEInfo.showPage`, the article page is started within the subarea. This does not have to be flexible all the time - but it may be on request. (Maybe there are several versions of displaying the detail data, depending on the article type).

When choosing the **Show Details** button, the method `showDetails()` is called. It prepares the adapter of the inner page to display the detail data of the requested article. Afterwards, the method `SUBCISPAGEInfo.refreshContentOfCurrentPage` is called in order to reload the embedded page. Consequently, the article details are shown.

See the JavaDoc documentation of class `SUBCISPAGEInfo`.

SUBCISPAGE2 Properties

Basic			
subcispageprop	Name of adapter property representing the control on server side. The property must be of type "TABSUBPAGESInfo". View the Java API Documentation for further information.	Optional	
width	Width of the control. There are three possibilities to define the width: (A) You do not define a width at all. In this case the width of the control will either be a default width or - in case of container controls - it will follow the width that is occupied by its content. (B) Pixel sizing: just input a number value (e.g. "100"). (C) Percentage sizing: input a percantage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a width this control can reference. If you specify this control to have a width of 50% then the parent element (e.g. an ITR-row) may itself define a width of	Optional	100 120 140 160 180 200 50% 100%

	"100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
width	(already explained above)		
height	(already explained above)		
borderwidth	Border size of control in pixels. Specify "0" not to render any border at all.	Optional	1 2 3 int-value
withownborder	Default is false. If WITHOWNBORDER is set to true, the subcispage2 control is rendered with its own 3D lookalike border. Set BORDERWIDTH to 0 if WITHOWNBORDER is set to true.	Optional	true false
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon.	Optional	

	Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.		
colspan	Column spanning of control. If you use TR table rows then you may sometimes want to control the number of columns your control occupies. By default it is "1" - but you may want to define the control to span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value
rowspan	Row spanning of control. If you use TR table rows then you may sometimes want to control the number of rows your control occupies. By default it is "1" - but you may want to define the control to span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value

10 ROWTABSUBPAGES Control

The ROWTABSUBPAGES control allows to switch between several Application Designer pages using tabs. The displayed number of tabs and names are derived dynamically from its adapter properties.

Optionally, the ROWTABSUBPAGES control may contain exactly one STRAIGHTTABPAGE as a subnode. STRAIGHTTABPAGE must be the first tab. This allows for combining ROWTABAREA behavior with ROWTABSUBPAGES behavior. Having a STRAIGHTTABPAGE as the first tab improves the loading behavior of ROWTABSUBPAGES. For an example, see the **80_straighttabpage** layout in the **cisdemos** project.
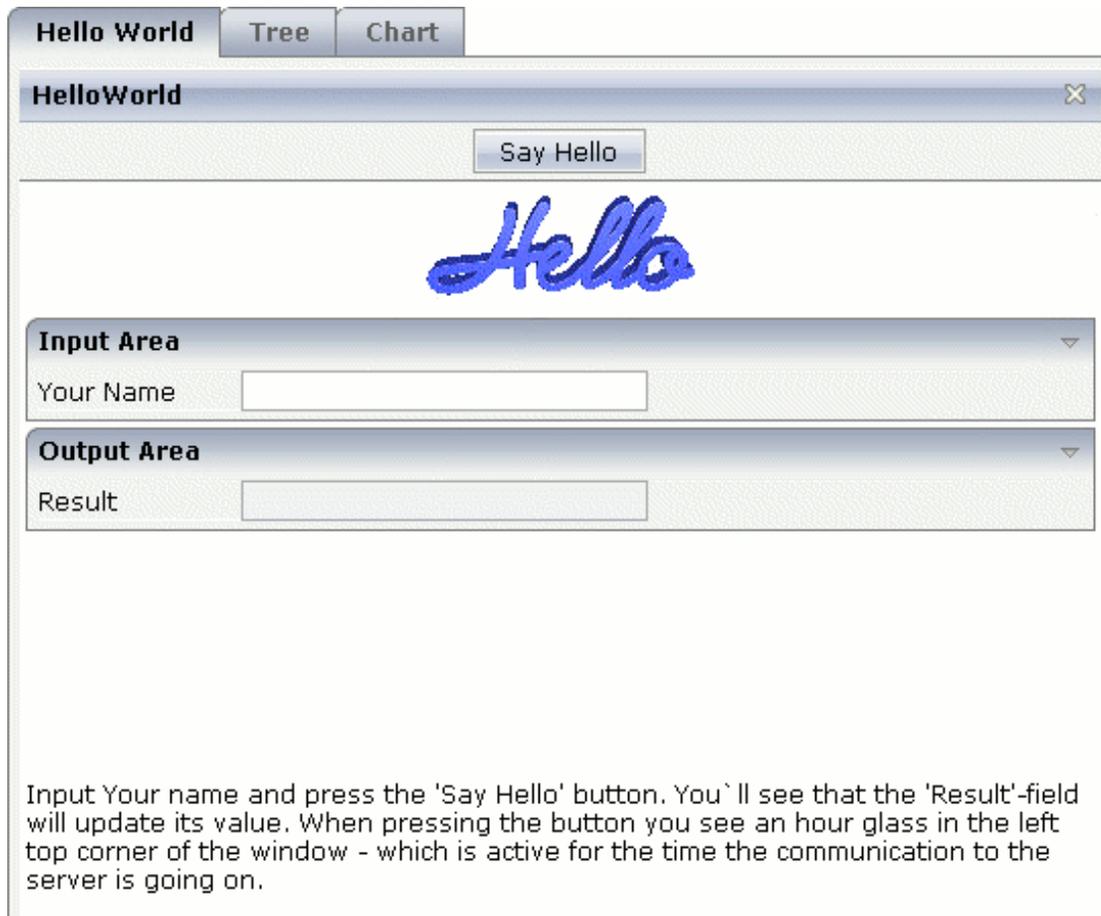
The XML definition is:

```
<pagebody>
    <rowtabsubpages pagesprop="tabpages" height="600" borderwidth="0">
    </rowtabsubpages>
</pagebody>
```

The ROWTABSUBPAGES control references a property `tabPages` which is provided by the adapter class of the page. The height can be specified depending on the whole page's height or can be fix. The page style can be manipulated directly.

The corresponding adapter source is:

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

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.util.TABSUBPAGESInfo;

public class RowTabSubPageAdapter
    extends Adapter
{
    // property >tabpages<
    TABSUBPAGESInfo m_tabPages = new TABSUBPAGESInfo();
    public TABSUBPAGESInfo getTabpages() { return m_tabPages; }

    /** initialisation - called when creating this instance*/
    public void init()
    {
        m_tabPages.addItem("Hello World","HelloWorld.html");
        m_tabPages.addItem("Tree","trees_01.html");
        //m_tabPages.addItem("Dynamic Combo", "HelloWorld.html");
        m_tabPages.addItem("Chart", "HelloWorld.html");
    }
}
```

The property `tabPages` is of type `com.softwareag.cis.server.util.TABSUBPAGESInfo`. There are methods for adding and removing items from the `tabPages` object. See the JavaDoc documentation. The number of items can be changed at any time.

Properties

Basic			
pagesprop	Name of adapter property representing the control on server side. The property must be of type "TABSUBPAGESInfo". View the Java API Documentation for further information.	Obligatory	
triggerserver	Flag indicating whether the adapter should be triggered if the user switches between pages. If set to true, method trigger() inside the TABSUBPAGESInfo object is called - before switching the page. Therefore the adapter can abort a page switch - maybe a user has to enter some data first on the current page before switching to another one.	Optional	true false
height	Height of the control. There are three possibilities to define the height:	Optional	100 150 200

	(A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percentage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.		250 300 250 400 50% 100%
scrollable	If set to "true" then small icons will appear on the right border of the control. If the size of the "tabs" is too big and some tabs are cut as consequence then you can use these icons for scrolling left and right.	Optional	true false
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
fastbufferswitch	If this property is switched to "true" (default is "false") then the contained subpages are buffered in a way that switching between tabs is not done by loading a new page but by just switching the visibility of pages. Please pay attention to that switching between pages in this case does not reload the page content from the server when switching! In order to enable fast switching you have to set the framebuffersize in cisconfig (n +1), n being the number of tabs to switch.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
borderwidth	Border width (in pixels) of the sub-page that is contained inside this control. Define "0" to avoid rendering any border.	Optional	1 2 3 int-value
leftindent	Inserts a horizontal distance left of the first "tab" and shifts the "tabs" to the right as consequence. The value you may define represents the number of pixels that are inserted.	Optional	1 2 3 int-value

paddingleft	Number of pixels which you want to keep as margin between the tab control's left border and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
paddingtop	Number of pixels which you want to keep as margin between the upper tab row and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
paddingright	Number of pixels which you want to keep as margin between the tab control's right border and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
paddingbottom	Number of pixels which you want to keep as margin between the bottom of the tab control and the inner sub page. Default is 5 pixel.	Optional	1 2 3 int-value
pagestyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame's source" function.	Optional	
Miscellaneous			
testtoolid	Use this attribute to assign a fixed control identifier that can be later on used within your test tool in order to do the object identification	Optional	

Performance Considerations

Many users like the subdivision of pages into "tabs". Application Designer offers several controls for this - let us compare the ROWTABSUBPAGES control described in this section with the ROWTABAREA control described in the *Layout Elements* documentation.

The ROWTABAREA control has certain content areas (TABPAGEs) and always makes one of them visible. This means: the page has much more HTML code and controls then are visible. The size of the page is important for the performance of the page in the browser: the bigger the size, the longer it takes the browser to render a page (also if it is already cached).

The ROWTABSUBPAGES control offers a subpage in which you can place contained pages.

Now imagine that you have 500 fields to be displayed inside "tabs": in this case, it is more performant to build one "mother page" containing the ROWTABSUBPAGES control and to have five "detail pages", one for each "tab", than having one big page with all 500 fields, arranged by a ROWTABAREA control.

In the demo workplace, there is an example in which you can "feel" the difference - please have a look!

11 Remark on Modularisation

This section describes one - important - technique for modularisation: embedding of pages into other pages.

This technique is useful for "rough granular" integration aspects: it is used to arrange pages with a dedicated task (e.g. the maintenance of an order) into other screens (e.g. an overview of all orders). Each page - both the "outer" page and the "inner" page - keep their "page behavior", i.e. they are talking independently to the server.

The hour glass icon indicates that a page is talking to its server adapter. If an "outer" page refreshes its "inner" page, the "outer" page first talks to the server, and afterwards the "inner" page. Therefore, there is more than one roundtrip between the client and the server.

As a consequence, it does not make sense (and it is not intended at all by Application Designer) to build up fine granular integration scenarios in which a group of fields is defined as an embeddable unit being used in several screens. This is the job of controls to easily build up your own one. Controls which you build render a group of controls (e.g. an area for entering an address) and can be re-used in different pages. Controls always talk to the server within the same roundtrip. They are available as design time controls - if you change the behavior of one control definition, all pages using this control have to be regenerated.

It is comparable with C programming. You have libraries that you put directly into your compilation process. If the libraries change, you have to recompile. This is the level of controls. On the other hand, you have units of rougher granularity: e.g. DLLs. These can be changed without letting your program know. This is the level of page integration.

III Multi Frame Pages

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

12 What are Multi Frame Pages?

Multi frame pages are a special set of pages. Normal pages represent a generated HTML page - a multi frame page represents a generated HTML frameset page.

A multi frame page does not contain controls but frames in which other pages are positioned. Each frame is associated with an ID (called "target" in this section). A frame may be:

- a normal HTML page
- an intelligent Application Designer page
- a frameset itself containing frames

Multi frame pages are the preferred way of arranging Application Designer pages in a frameset. Besides enhanced possibilities of communication between frames, multi frame pages automatically take care of keeping all Application Designer frames inside the same session. See section *Session Management* for more details.

13   **Definition of Multi Frame Pages**

The definition of multi frame pages is done with the Layout Painter. When you create a new layout, a dialog appears in which you select a template. To create a multi frame page, you have to select the "Multi Frame Page" template. The Layout Painter will open just as usual, but instead of having the PAGE control as the highest control, you now see the control MFPAGE. You can reach a number of controls that are related to multi frame page management.

The following controls are "normal frame controls" (they are described below):

- MFPAGE - the top element of multi frame pages.

- MFCISFRAME - a frame in which an Application Designer HTML page is loaded.

- MFHTMLFRAME - a frame in which a normal HTML page is loaded.

- MFFRAMESET - an area that can be subdivided into frames itself.

The following controls are "workplace controls" (they are described in the section *Application Designer Workplace Framework*. The Application Designer workplace - which is described in the *Development Workplace* documentation - is based on these controls.

- MFWPFUNCTIONS

- MFWPACTIVEFUNCTIONS

- MFWPCONTENT

MFPAGE

The MFPAGE is the top node of every multi frame page. It can be subdivided into frames or framesets.

Basic			
separation	Specifies how the corresponding internally used frameset is subdivided: choose "rows" for subdividing into rows, "cols" for subdividing into columns.	Obligatory	rows cols
sizing	Defines the size of the contained sub-frames. If you have three sub-frames to show up inside the page then you might specify "200,200,*" to specify how the height (if SEPARATION is "rows") or the width (if SEPARATION is "cols") is distributed among the frames. You can speficy per frame either a pixel value or a "*".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			

border	Space between frames contained in the frameset that is internally built up.	Optional	1 2 3 int-value
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
frameborder	Defines if to display a border around the contained frames. Valid values are "true" or "false".	Optional	true false
framespacing	Defines the amount of additional space between the frames. Value is a pixel value.	Optional	1 2 3 int-value
framesetstyle	Style passed to the HTML-frameset definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

MFCISFRAME

The MFCISFRAME represents a frame in which an Application Designer page is shown. The name of the page is passed as a parameter.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specifiy with the TARGET property.	Obligatory	
cisurl	URL of the page to be shown inside. Use /project/page.html as syntax, e.g. "/HTMLBasedGUI/empty.html". Do NOT use only page.html believing that you do not have to specify the project because the multi frame page runs in the same project than the page you want to open - you ALWAYS have to specify the project!	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence. Valid values are "true" and "false". Default is "true".	Optional	true false
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080

			#000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
withownborder	Flag that indicates if started pages show an own border. Default is false.	Optional	true false
Unload Behaviour			
unloadbehaviour	Reaction that CIS should take if the page inside the frame is closed. Possible values are "NOTHING" for doing nothing and "REMOVESESSION" for removing the session on server side. Do not define this property just "by accident" but leave it to the default ("NOTHING"). You only switch to "REMOVESESSION" if you want that the server side session is destroyed when leaving the page. This is the case if you have one page that clearly indicates the closing of a session at the point of time when the page is closed.	Optional	NOTHING REMOVESESSION

Applications can change the page that is shown inside the MFCISFRAME by using the method `Adapter.openCISPageInTarget(...)`.

MFHTMLFRAME

The MFHTMLFRAME represents a frame in which a normal HTML page is shown. This page can be a static HTML page or any URL - e.g. a URL referring to a certain JSP page.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specifiy with the TARGET property.	Obligatory	
url	URL to be opened inside the frame. The URL can be defined relative to the multi frame page or can be defined in an absolute way.. Example: You can define "../HTMLBasedGUI/workplace/header2.html" - or "http://www.softwareag.com".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence. Valid values are "true" and "false". Default is "true".	Optional	true false
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false
scrolling	Boolean that indicates whether the frame can be scrolled. Default is true.	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

bordercolor	Sets the border color of the frame set.	Optional	#FF0000
			#00FF00
			#0000FF
			#FFFFFF
			#808080
			#000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1
			2
			3
			int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1
			2
			3
			int-value

MFFRAMESET

The MFFRAMESET represents a frame that is internally again divided into frames. The MF-FRAMESET definition decides whether to divide into rows or columns, and how to size the inner frames.

Basic			
target	Id of the frame. Must be unique inside the frameset page. Must only contain alphanumeric characters. The id is important! CIS offers certain methods inside the Model-class that allow an adapter to start operations for a certain frame (e.g. openeCIPageInFrame(...)). As part of the parameters of these methods a target-id is passed. The target-id is exactly the id you specifiy with the TARGET property.	Obligatory	
separation	Specifies how the corresponding internally used frameset is subdivided: choose "rows" for subdividing into rows, "cols" for subdividing into columns.	Obligatory	rows cols

sizing	Defines the size of the contained sub-frames. If you have three sub-frames to show up inside the page then you might specify "200,200,*" to specify how the height (if SEPARATION is "rows") or the width (if SEPARATION is "cols") is distributed among the frames. You can speficy per frame either a pixel value or a "*".	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
border	Space between frames contained in the frameset that is internally built up.	Optional	1 2 3 int-value
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
frameborder	Defines if to display a border around the contained frames. Valid values are "true" or "false".	Optional	true false
framespacing	Defines the amount of additional space between the frames. Value is a pixel value.	Optional	1 2 3 int-value
framesetstyle	Style passed to the HTML-frameset definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold

14   Example

The example that will be built in this section produces the following output:
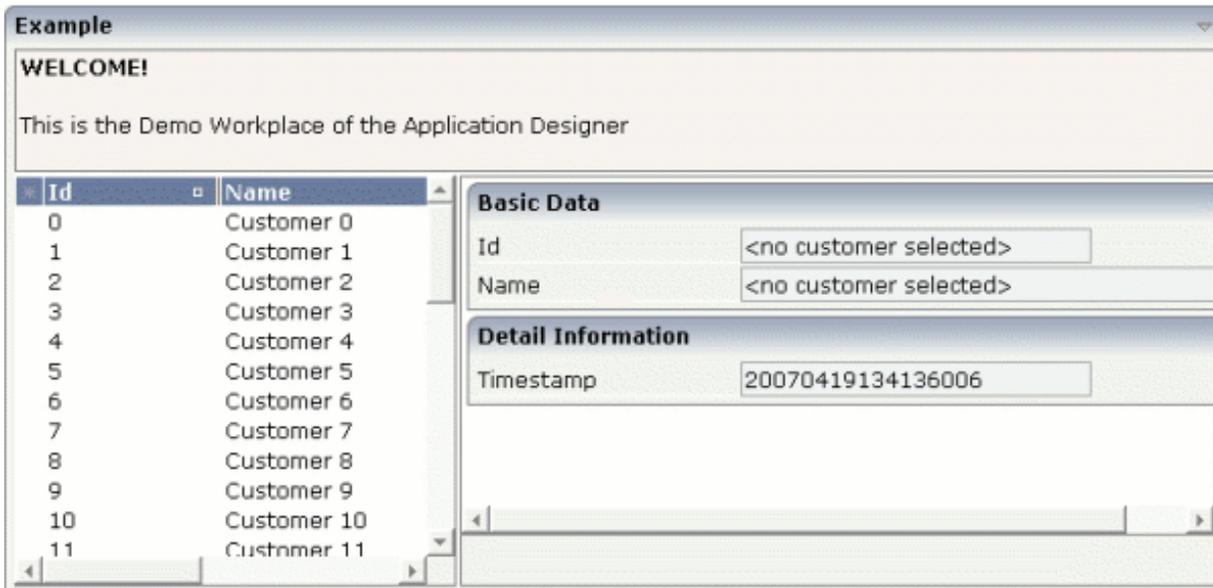
When selecting a customer on the left, the customer detail screen is displayed on the right:
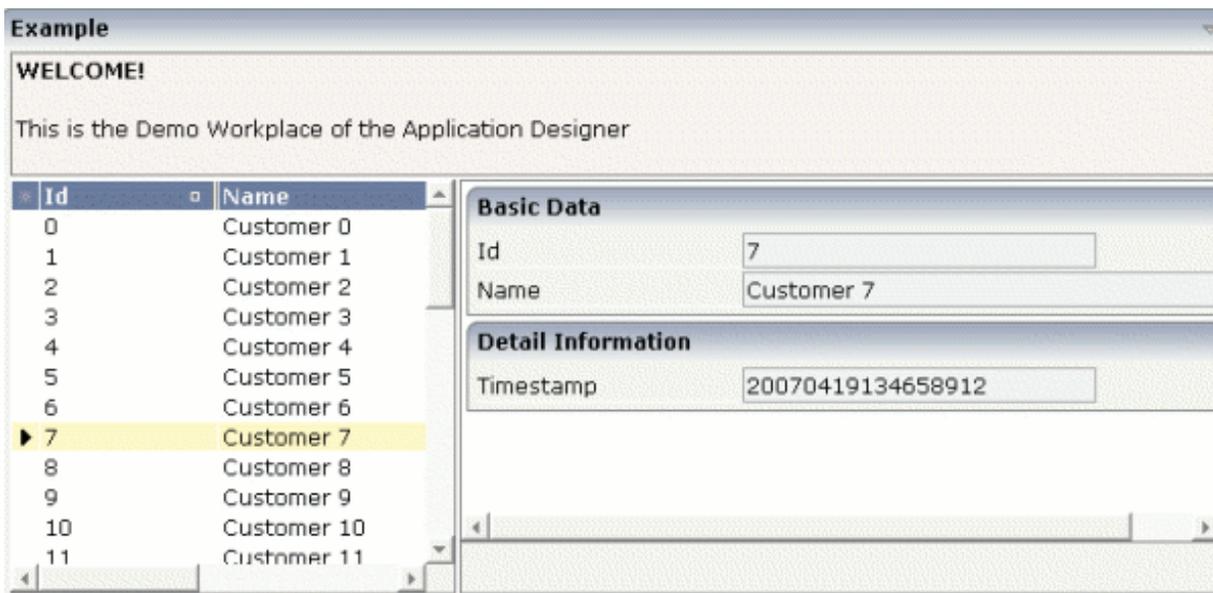
When the user selects another record on the left, the screen on the right is updated accordingly.

The Multi Frame Page Around

First let us have a look at the multi frame page itself. The layout definition is as follows:

```
<mfpage separation="rows" sizing="70,*">
    <mfhtmlframe target="HEADER"
                 url="../HTMLBasedGUI/workplace/welcome.html"
                 resizable="true"
                 withborder="false"
                 scrolling="false"
                 framestyle="border: 1px #808080 solid">
    </mfhtmlframe>
    <mfframeset target="AROUND"
                separation="cols"
                sizing="200,*">
        <mfcisframe target="INNERLEFT"
                    cisurl="/cisdemos/25_mfinnerleft.html"
                    framestyle="border-right: 1px solid #808080;
                    border-bottom: 1px solid #808080">
        </mfcisframe>
        <mfcisframe target="INNERRIGHT"
                    cisurl="/HTMLBasedGUI/empty.html"
                    framestyle="border: 1px solid #808080">
        </mfcisframe>
    </mfframeset>
</mfpage
```

The page is subdivided into three frames: "HEADER", "INNERLEFT" and "INNERRIGHT". Two of them are Application Designer frames, one is an HTML frame. Every frame is pointing to a certain page.

The Left Frame

The INNERLEFT frame's page displays a text grid and lets the user select from the list of items. The layout definition is:

```
<page model="MFInnerLeftAdapter">
    <pagebody horizdist="false" takefullheight="true">
        <itr height="100%" fixlayout="true" width="100%">
            <textgrid2 griddataprop="customers" width="100%" height="100%" ↵
selectprop="selected"
                       singleselect="true" hscroll="true" ↵
directselectmethod="onSelect"
                       directselectevent="onclick">
                <column name="Id" property="id" width="100">
```

```
                    </column>
                    <column name="Name" property="name" width="400">
                    </column>
                </textgrid2>
            </itr>
        </pagebody>
</page>
```

The adapter implementation is done in the following way:

```java
import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.ServerLog;
import com.softwareag.cis.server.util.TEXTGRIDCollection;

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

 public class CustomerInfo
 {
            boolean m_selected;
            String m_id;
            String m_name;
            public String getId() { return m_id; }
            public String getName() { return m_name; }
            public boolean getSelected() { return m_selected; }
            public void setId(String string) { m_id = string; }
            public void setName(String string) { m_name = string; }
            public void setSelected(boolean b) { m_selected = b; }
 }
 // -------------------------------------------------------------------------
 // members
 // -------------------------------------------------------------------------

 TEXTGRIDCollection m_customers = new TEXTGRIDCollection();

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

 public TEXTGRIDCollection getCustomers() { return m_customers; }

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

 public void init()
 {
  super.init();
```

```
  for (int i=0; i<40; i++)
  {
   CustomerInfo info = new CustomerInfo();
   ci.setId(""+i);
   ci.setName("Customer " + i);
   m_customers.add(ci);
  }
}

public void onSelect()
{
 try
 {
  CustomerInfo info = (CustomerInfo)m_customers.findLastSelectedItem();
  // prepare adapter of right frame
  MFInnerRightAdapter mfira =
                    (MFInnerRightAdapter)findAdapter(MFInnerRightAdapter.class);

  mfira.prepare(ci.getId());
  // preload adapter so that only one request is executed
  includeAdapterInResponse("../_DevelopersGuide/mfinnerright.html",false);
  // refersh target
  refreshTarget("INNERRIGHT");
 }
 catch (Throwable t) { ServerLog.appendException(t); }
 }
}
```

The class contains the following:

- An inner class for the text grid items.

- An `init` method for filling the text grid.

- A `onSelect()` method that is called when the user selects a text grid line.

The "critical" lines of code are inside the `onSelect()` method. Inside the method

- the selected line is determined,

- the adapter of the right neighbor screen is prepared so that it shows the data of the selected line,

- the right page is switched to the detail page (if first call) or

- the right page is refreshed to present the correct adapter information.

The Right Frame

The right frame is loaded with *HTMLBasedGUI/empty.html* first. With the first selection in the text grid, the detail page is opened inside the right frame. Afterwards, the detail page is refreshed to update its content.

15 Communication between Frames

You already saw some methods in the **previous** section enabling one frame to open pages in another frame and to refresh information of other frames.

API inside the Adapter Class

The following table shows a summary of functions that you can reach in your adapter class which inherits from `com.softwareag.cis.server.Adapter`. See the JavaDoc documentation for implementation details.

Method	Description
openCISPageInTarget(...)	Opens a certain Application Designer HTML page inside a certain frame which is identified by its target ID. There is a set of methods with different parameter notation. The default method just needs to know the page URL and the ID of the frame. Other methods expect more information, e.g. if you want to open the Application Designer page in a different subsession.
refreshTarget(...)	Refreshes the target's frame content. This method is to be used if you want the target frame not to change its page but to update its content. In the **example** in the previous section, this method is used after having updated the right frame's adapter on the server side.
invokeMethodInTarget(...)	Invokes a method in the target frame's Application Designer HTML page. The call is triggered from the client - for example, imagine that a button supporting the method is pressed in the target frame's Application Designer HTML page.
sizeTarget(...)	Manipulate the size of the target. Each target gets a certain size by the frame set definition on top of it (e.g. if the frame set definition has a sizing of "200,300,*", then the second frame has a size of "300". You can change the size of a target by using this method.

Pay Attention to Request Processing

Be aware of the request processing in the browser: only the page which sends a request (e.g. the left page in the **example**) is the active page and will process the response. All other pages living in neighboring frames are by default not affected.

Consequence: if you want to change or refresh these pages, you have to explicitly do so using the API presented in one of the previous sections.

⚠️ **Important:** The adapter that processes the request is the one to call the API methods.

Session Management (I)

Maybe you have already tried to build multi frame pages on your own, using HTML framesets:

```
...
...
<frameset cols="*,*">
  <frame src="../servlet/StartCISPage?PAGEURL=/project/left.html">
  <frame src="../servlet/StartCISPage?PAGEURL=/project/right.html">
</frameset>
...
...
```

If so, you will have seen that in each of the frames, the Application Designer page will be opened correctly. However, both pages are running in independent sessions (not subsessions).

Opening the same pages using Application Designer's MF* controls (MFFRAMESET, MFCISPAGE) will keep both pages inisde the same session and subsession.

> **Note:** Details on session management are provided in the section *Session Management*.

Session Management (II)

When communicating between frames, e.g. by using the method `Adapter.openCISPageInTarget()`, the default is that the page that is opened in another target will be opened in the same session/subsession as the one that initiated the frame communication. Session ID and subsession ID are taken over by default.

There are certain variants of the `openCISPageInTarget()` method that allow to control the management of a subsession in a more fine granular way: you may pass as parameter the ID of the subsession in which a page should be opened in another page; i.e. you can explicitly decouple the other frame's subsession from your own one.

The workplace that comes with Application Designer makes use of this: every time you open a content window, this content is managed in its own subsession, being decoupled from the workplace's subsessions and being decoupled from other content pages' subsessions.

Use these functions with care: typically all application adapters should run in one subsession, and only an "outside function" (such as the workplace management) should take care of starting various contents in various subsessions.

16 Combination with Normal Application Designer Pages

There is no problem to integrate multi frame pages into other Application Designer pages. The mechanisms described in the section *Embedding Pages into Pages* are valid for both normal Application Designer pages and multi frame Application Designer pages.

This means:

- You can embed multi frame pages into normal Application Designer pages via the SUBCISPAGE2 control.
- You can embed multi frame pages into normal Application Designer pages via the ROWTAB-SUBPAGES control.

IV Embedding Pages into a Workplace

In the *First Steps*, you learned already how to build pages that are generated by the Layout Painter. This part explains how to integrate Application Designer pages into workplace/portal environments. There are different scenarios that are described here:

- Usage of the Application Designer workplace framework that lets you design and implement individual workplaces for your application.

- Integrating Application Designer pages into various portal scenarios by opening them with a URL.

- Writing a workplace framework on your own - i.e. use Application Designer in order to build your workplace, but not on base of the Application Designer workplace framework.

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

Application Designer Workplace Framework
Integration into Other Workplace/Portal Scenarios
Extended Functions in the Application Designer Workplace
Building Own Workplaces as a Frameset Definition

17 Application Designer Workplace Framework

The demo workplace (as well as the IDE workplace) provides examples of workplaces built on base of this framework.
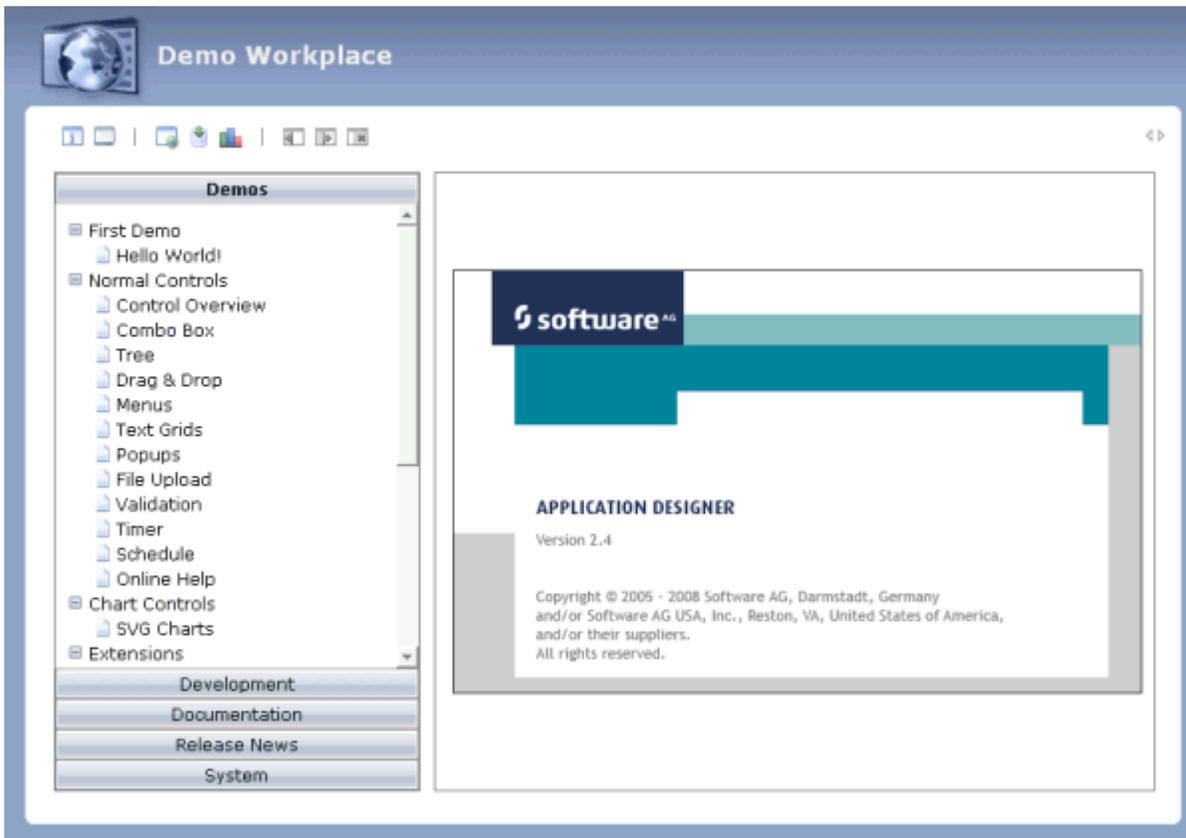
The workplace framework bases on the multi frame page management described in the previous part. It offers the following:

- flexible arrangement of frames,
- predefined frames containing workplace logic,
- dynamic loading of available functions,
- possibility to change the environment at runtime via the Java API,
- execution of multiple tasks between which the user can switch ("multi document interface").

Framework Overview

An Application Designer workplace is a certain arrangement of frames in a multi frame page. Some of the frames have predefined tasks. Have a look at the demo workplace in which you can already see the most important frames:
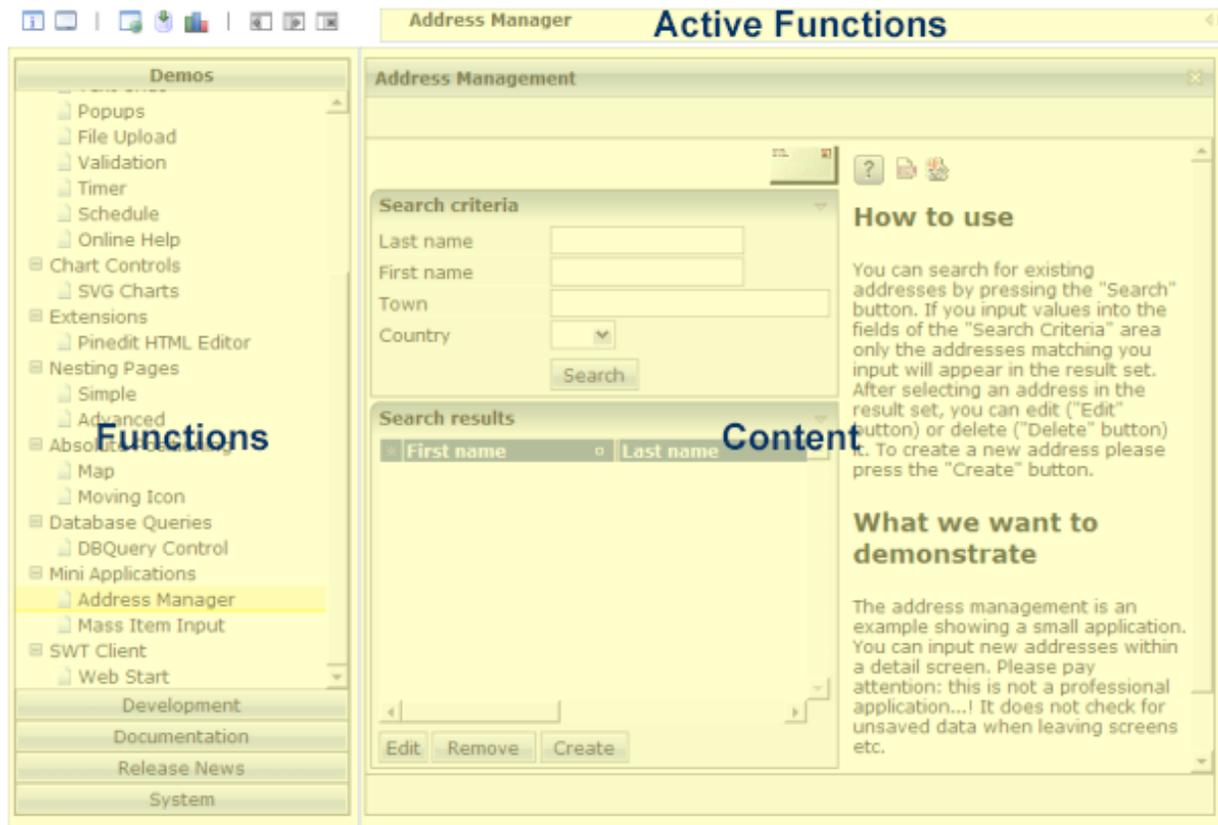
The "Functions" frame contains the available functions that can be chosen and invoked by the user. The "Content" frame contains the page or page sequence that is opened if a function is selected. The "Active Functions" frame shows the functions that were opened by the user and allows the user to navigate between the active functions.

Have a look at the XML layout definitions for this workplace; it consists of an inner definition in which the main frames are arranged and an outer definition that adds some additional decoration around. The inner page (*com.softwareag.cis.workplace.MFInner.xml*) is:

```
<mfpage separation="rows" sizing="20,*">
    <mfwpactivefunctions resizable="false" withborder="false" scrolling="false"
                         framestyle="border: 0px solid #000000">
    </mfwpactivefunctions>
    <mfframeset target="ZZZ" separation="cols" sizing="265,*">
        <mfframeset target="LEFTPART" separation="rows" sizing="*,87" border="true"
                    framesetstyle="border: 1px solid #808080">
            <mfwpfunctions ↵
bootstrapclass="com.softwareag.cis.workplace.MFDefaultBootstrapInfoProvider"
                           serversidescrolling="false" framestyle="border: 1 solid ↵
#808080;">
            </mfwpfunctions>
            <mfhtmlframe target="NEWS" url="../HTMLBasedGUI/workplace/welcome.html"
                         resizable="true" withborder="false" scrolling="true"
                         framestyle="border: 1px solid #808080">
            </mfhtmlframe>
        </mfframeset>
        <mfwpcontent resizable="true" withborder="true" scrolling="false"
                     framestyle="border: 1 solid #808080;">
        </mfwpcontent>
    </mfframeset>
</mfpage>
```

You see that there are three special frame controls that are used internally: MFWPFUNCTIONS, MFWPACTIVEFUNCTIONS and MFWPCONTENT. In addition, there is one HTML page arranged below the MFWPFUNCTIONS control.

Let us take a closer look at each of the three workplace frame controls.

Functions Frame: MFWPFUNCTIONS

This is the frame to hold the available functions to be selected by the user. The control has the following properties:

Basic		
bootstrapclass	Name of the class that is responsible for passing the initial workplace configuration. The class must support interface "IMFWorkplace2" and must support a constructor without parameters. When being displayed the workplace creates an instance of this class and asks for an object that represents the workplace setup. Have a look into the javadoc-documentation for interface "IMFWorkplace2" for more information.	Optional
bootstrapinfourl	URL to an .xml file that holds the initial workplace configuration. Do not use BOOTSTRAPINFOURL and BOOSTRAPCLASS at the same time!	Optional

	Use /project/directory/doc.xml as syntax, e.g. /HTMLBasedGUI/workplace/bootstrapworkplaceinfo.xml.		
serversidescrolling	Flag that decides if the function tree providing the available workplaces functions support client side scrolling (default, "false") or supports server side scrolling ("true"). Server side scrolling should be used if a function tree containes more than 100 nodes.	Optional	true false
defaultcontentpage	URL of a page that is shown in the 'content area' by default.	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
contentstylesheet	Style sheet that should be used for the content that is started inside the workplace.	Optional	
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value

activefunctionsvariant	Defines how the MFWPACTIVEFUNCTIONS frame displays the list of started pages. You can either use a STRIPSEL or TABSTRIP control. Default is "tabstrip".	Optional	tabstrip stripsel
withownborder	Flag that indicates if the functions page shows an additional border. Default is false.	Optional	true false
workplacestylesheet	Style sheet that should be used for the workplace itself.	Optional	
withplusminus	If set to "true" then +/- Icons will be rendered in front of the mfwpfuntions.	Optional	true false
workplaceproject	If set to a valid project name, standard messages and standard dialogs used by the default workplace framework will be generated into this project. At runtime the messages and dialogs of this project will be used instead of the default ones of the HTMLBasedGUI project. Generated multilanguage files:workplace and popups. Generated layouts:popupyesno and popupok.	Optional	

Active Functions Frame: MFWPACTIVEFUNCTIONS

This frame shows the functions that the user started and between which the user can switch.

Basic			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence. Valid values are "true" and "false". Default is "true".	Optional	true false
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false
scrolling	Boolean that indicates whether the frame can be scrolled. Default is true.	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF

			font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Content Frame: MFWPCONTENT

This is the frame in which content is started that is selected from the functions area.

Basic			
resizable	Decision if the user is able to resize the frame. This property must be in synch with the definition in the "neighbour frames". If the neighbour frames do not support resizing then it will not be offered to the user as consequence. Valid values are "true" and "false". Default is "true".	Optional	true false
withborder	Boolean value defining if the frame has a border on its own. Default is "false".	Optional	true false

scrolling	Boolean that indicates whether the frame can be scrolled. Default is true.	Optional	true false
framestyle	Style that is passed to the HTML-FRAME definition that is internally generated.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
bordercolor	Sets the border color of the frame set.	Optional	#FF0000 #00FF00 #0000FF #FFFFFF #808080 #000000
marginheight	Defines top and bottom margin height. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
marginwidth	Defines left and right margin width. Value is a pixel value. Default is "0".	Optional	1 2 3 int-value
withownborder	Flag that indicates if started pages show an own border. Default is false.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	

Filling the MFWPFUNCTIONS Frame

The MFWPFUNCTIONS frame itself connects to an instance of the class that is named inside the `bootstrapclass` property. This class must support a constructor without parameters and must support an interface `com.softwareag.cis.workplace.IMFWorkplaceBootstrapInfoProvider2`:

```
public interface IMFWorkplaceBootstrapInfoProvider2
{
    public MFWorkplaceInfo getWorkplaceInfo(IMFWorkplaceBootstrapInfo envInfo);
}
```

The interface contains one method `getWorkplaceInfo(...)` that returns an object of type `MFWorkplaceInfo`. Inside the `MFWorkplaceInfo` object, the logical structure of the functions that are offered to the user is defined.

The MFWPFUNCTIONS frame consists of certain subcomponents:
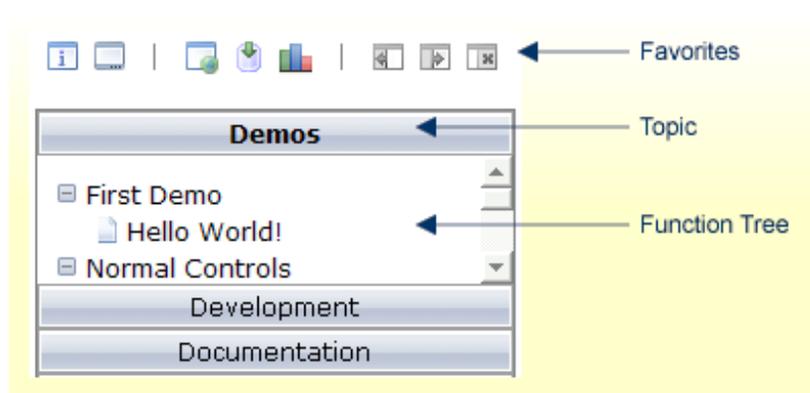
Each topic holds one function tree. The tree is opened when the user chooses the corresponding button. The tree contains nodes; each node is associated with a certain function, e.g. a node may start a page inside the content area of the workplace. Some nodes may be selected as favorites and are shown in a favorite bar.

The `MFWorkplaceInfo` object that is required by the interface definition above is the logical reflection of this structure. The following code shows the code for setting up the demo workplace:

```
public class MFDefaultBootstrapInfoProvider
 implements IMFWorkplaceBootstrapInfoProvider2,
            MFWorkplaceConstants
{
    // -------------------------------------------------------------------------
    // public access
    // -------------------------------------------------------------------------

    /** */
```

```
    public MFWorkplaceInfo getWorkplaceInfo(IMFWorkplaceBootstrapInfo envInfo)
    {
        MFWorkplaceTopic topic;
        MFWorkplaceTreeNodeFolder topNode;
        TREECollection tc;

        MFWorkplaceInfo workplaceInfo = new ↵
MFWorkplaceInfo("/HTMLBasedGUI/empty.html",

                                                             ↵
"../softwareag/styles/CIS_DEFAULT.css");

        // ------------------------------------------------------------------------
        // Demo topic
        // ------------------------------------------------------------------------

        topic = new MFWorkplaceTopic("Demos",workplaceInfo);
        tc = topic.getTree();

        topNode = new MFWorkplaceTreeNodeFolder("First Demo");;
        topNode.setOpened(TREECollection.ST_OPENED);
        tc.addTopNode(topNode,false);


        MFWorkplaceTreeNodeCISPage helloWorldNode =
            new MFWorkplaceTreeNodeCISPage("Hello World!",
            "/cisdemos/DEMO_HelloWorld.html",true,true);
        tc.addSubNode(helloWorldNode,topNode,true,false);
        workplaceInfo.addFavourite(helloWorldNode,"images/fav_hello.gif");

        topNode = new MFWorkplaceTreeNodeFolder("Normal Controls");
        topNode.setOpened(TREECollection.ST_OPENED);
        tc.addTopNode(topNode,false);

        tc.addSubNode(new MFWorkplaceTreeNodeCISPage("Control Overview",
                        ↵
"/cisdemos/DEMO_ControlOverview.html",true,true),topNode,true,false);
        tc.addSubNode(new MFWorkplaceTreeNodeCISPage("Combo Box",
                    "/cisdemos/DEMO_ComboDyn.html",true,true),topNode,true,false); ↵
    }

        ...
        ...
        ...
        ...

        // ------------------------------------------------------------------------
        // Development topic
        // ------------------------------------------------------------------------

        topic = new MFWorkplaceTopic("Development",workplaceInfo);
        tc = topic.getTree();
```

```
        topNode = new MFWorkplaceTreeNodeFolder("Layout");
        topNode.setOpened(TREECollection.ST_OPENED);
        tc.addTopNode(topNode,false);

        tc.addSubNode(new MFWorkplaceTreeNodeCISPage("Project Manager",
                    ↵
"/HTMLBasedGUI/com.softwareag.cis.editor.projectmgr.html",true,true),
                    topNode,true,false);

        MFWorkplaceTreeNodeCISPage layoutNode =
            new MFWorkplaceTreeNodeCISPage("Layout Manager",
          "/HTMLBasedGUI/com.softwareag.cis.editor.editorgenerate.html",true,true);
        tc.addSubNode(layoutNode,topNode,true,false);

        workplaceInfo.addFavourite(layoutNode,"DISTANCE");
        workplaceInfo.addFavourite(layoutNode,"images/fav_layoutpainter.gif");


        ...
        ...
        ...
        ...


        return workplaceInfo;
    }
}
```

See the Javadoc API documentation for more details on the API.

Tree Node Types

There are different types of tree nodes that you place inside a topic's tree. In the example above, you already saw two tree node types: MFWorkplaceTreeNodeFolder and MFWorkplaceTreeNodeCISPage. The complete list of tree node types is:

Type	Description
MFWorkplaceTreeNodeFolder	A folder in the tree. Has no further functions.
MFWorkplaceTreeNodeCISPage	A node that opens an Application Designer page in the content area.
MFWorkplaceTreeNodeHTMLPage	A node that opens a normal URL in the content area.
MFWorkplaceTreeNodeCISPopup	A node that starts an Application Designer page inside a pop-up.
MFWorkplaceTreeNodeHTMLPopup	A node that starts a normal URL inside a pop-up.
MFWorkplaceTreeNodeCISTarget	A node that starts an Application Designer page inside a named target frame that is part of the workplace multi frame page.
MFWorkplaceTreeNodeHTMLTarget	A node that starts a normal URL inside a named target frame that is part of the workplace multi frame page.

Type	Description
MFWorkplaceTreeNodeCallback	A node that invokes a "dark" API in order to just call a function without visual output. The function may, for example, modify the workplace content.

A detailed description of the Java API can be found in the Javadoc API documentation.

Filling the MFWPFUNCTIONS Frame without any Java Coding: MFWPBOOT-STRAPINFO

There is also the possibility to fill the MFWPFUNCTIONS frame without any Java coding by using the `bootstrapinfourl` property. This property expects an URL to an XML file that represents the workplace setup (for example, *HTMLBasedGUI\workplace\defaultbootstrapinfo.xml*).

Have a look at the corresponding XML file:

```
<mfwpbootstrapinfo
        defaultcontentpage="/HTMLBasedGUI/empty.html"
        workplacestylesheet="../cis/styles/CIS_DEFAULT.css"
        synchtabnavigation="true"
        showdustbin="true"
        withtakeouttopopup="false"
        withcloseallwindowsicon="false"
        ↵
mfworkplaceeventlistener="com.softwareag.cis.workplace.MFDefaultEventListener"
        targetnameofresizableleftpart="AVAILABLEACTIVITIES"
        translationproject="cisdemos"
        translationreference="mfworkplace">

<!-- Start Topic 'Demos'-->
  <mfwptopic
        name="Demos"
        textid="topic.demos"
        treeclass="WORKPLACETOPIC1ClientTree">

<!--TREE Begin First Demo -->
    <mfwpfolder
        name="First Demo"
        draginfo="First Demo"
        opened="true">

     <mfwpopencispage
        name="Hello World!"
        activityurl="/cisdemos/DEMO_HelloWorld.html"
        onlyoneinstance="true"
        followpageswitches="true"
        draginfo="DEMO_HelloWorld">
```

```
      </mfwpopencispage>

   </mfwpfolder>
<!--TREE End First Demo -->

...

<!-- End Topic 'Demos'-->
  </mfwptopic>

...

</mfwpbootstrapinfo>
```

> **Note:** To make sure that you are using a proper *bootstrapinfo.xml* file, use the XML Schema *editor.xsd* (and all corresponding XSD files) to validate your XML file (for example, in XMLSpy).

Overview of the bootstrapinfo hierarchy:

```
<mfwpbootstrapinfo>          // root tag
    <mfwptopic>              // new topic
       <mfwpfolder>          // MFWorkplaceTreeNodeFolder
       <mfwpopencispage>     // MFWorkplaceTreeNodeCISPage
       <mfwpopencispopup>    // MFWorkplaceTreeNodeCISPopup
       <mfwpopencistarget>   // MFWorkplaceTreeNodeCISTarget
       <mfwpcallback>        // MFWorkplaceTreeNodeCallback
       <mfwpopenhtmlpage>    // MFWorkplaceTreeNodeHTMLPage
       <mfwpopenhtmlpopup>   // MFWorkplaceTreeNodeHTMLPopup
       <mfwpopenhtmltarget>  // MFWorkplaceTreeNodeHTMLTarget
```

`<mfwpfolder>` can contain each of the other `<mfwptopic>` subtags including itself.

The following topics are covered below:

- MFWPBOOTSTRAPINFO Properties
- MFWPTOPIC Properties
- MFWPFOLDER Properties
- MFWPOPENCISPAGE Properties
- MFWPOPENCISPOPUP Properties
- MFWPOPENCISTARGET Properties
- MFWPCALLBACK Properties
- MFWPOPENHTMLPAGE Properties
- MFWPOPENHTMLPOPUP Properties

- MFWPOPENHTMLTARGET Properties

MFWPBOOTSTRAPINFO Properties

Basic			
defaultcontentpage	The workplace consists out of several frames, one of it the content frame. If there is no active activity in the workplace then the defaultContentPage is displayed inside the content frame. You can use this in two ways: (1) Either create one "background page" which always is shown in an "empty" workplace. (2) Or create one "background page" which the workplace opens by default. E.g. you want in a start-workplace to first present to the user a logon page. EXAMPLE: "/HTMLBasedGUI/empty.html"	Optional	
workplacestylesheet	The stlye sheet which is used for the left and top frame of the workplace. If no style sheet is specified then the workplace adapts to the standard style sheet which is kept in the CISsession context. You typically want to use one fix child for a workplace - because the workplace is typically embedded in some other frames arranging some graphics/etc. around, and you do not want the workplace colour's to change independent from this. EXAMPLE: "/cis/styles/XYZ_STLYE.css"	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
translationproject	Name of the project where the actual used multilanguage file is located. e.g. cisdemos	Optional	
translationreference	Name of the multilanguage .csv file. e.g. test (if the file test.csv should be used)	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
mfworkplaceeventlistener	Use this interface to react on workplace events. (1) Create an implementation of this interface (2) Use method `MFWorkplaceInfo.registerMFWorkplaceEventListener` to register your class	Optional	

	(3) Use method `NODEInfo.setDropInfo` on each tree item to be able to drag that item Step two and three are typically done within the "bootstrap info provider"-class A CISworkplace is a certain arrangement of frames in a multi frame page. The "functions"-frame (MFWPFUNCTIONS) holds the available functions to be selected by the user (click with the left mouse Button). In addition you can provide for right mouse button menu or drag and drop within the function tree. With that you may allow users to add/remove/shift menu items (personalization).		
targetnameofresizableleftpart	The workplace may contain a favourite list. At the bottom of the favourite list there are some items by which you can influence the size of the corresponding left part of the workplace. The name of the target frame to be resized is passed with this method.	Optional	
View			
showdustbin	Flag that indicates wether the dustbin (have a look at the DEMO WORKPLACE) is shown or not. Boolean value, default is false.	Optional	true false
synchtabnavigation	Set flag that decides if the tree "on the left" is synchronized with the tab navigation "on the top". If the user selects an opened activity in the tab strip then the corresponsding tree node and topic is shown as consequence. Pay attention: the base of the synchronization is the naming of nodes. There is currently no naming concept beyond (that e.g. assigns ids to nodes). Make sure, your tree nodes are set in a way that each one holds a unique name. Use the tabText (setTabText) in order to make nodes unique! true ==> synchronization is done; false ==> synchronization is not done; default is false.	Optional	true false
withcloseallwindowsicon	Flag that indicates whether the CloseAllWindowsIcon is shown in the workplace or not. Boolean value, default is false.	Optional	true false
withtakeouttopopup	Flag that indicates	Optional	true false
browsertitleappendix	Customize the browser title. An empty string means, that you don't want to set the activity title as browser title. A non-empty string means, that for each activity the browser title is set to a	Optional	

	concatenated value of the activity title and the browsertitleappendix you specified. Example: My Activity - My Browser Title Appendix.		
helpframewidth	The width of the cis help frame in pixels. Use this property in case you are using the CISHELP frame in your workplace for showing online help pages.	Optional	100 120 140 160 180 200 50% 100%

MFWPTOPIC Properties

Basic			
name	Text of the topic.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
buttonstyle	Style info that is passed to the button representing the topic.	Optional	
iconurl	The button that represents this topic may have an additional icon in front of the text. Use this parameter to set the icon URL.	Optional	
treestyle	Background style for the tree. You can e.g. define background colors and background pictures. Avoid the usage of ' and " characters. Please also have a look onto the method "setStyleClass" - via this method you can pass a reference to a CSS class.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
treeclass	Sets the style class for rendering the tree area of the topic. There are 10 standard style classes available in the default style sheet: PLACETOPIC1ClientTree to WORKPLACETOPIC10ClientTree. These style sheets can be maintained within the CISstyle sheet editor.	Optional	

| tooltip | Tooltip of the node. | Optional | |
| tooltipid | Text ID of the tooltip. | Optional | |

MFWPFOLDER Properties

Basic			
name	Text of the tree node folder.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
opened	Flag that indicates whether the folder is opened or not. Boolean value	Optional	true false
tooltip	Text of the tooltip of the tree node folder.	Optional	
tooltipid	Text ID of the tooltip.	Optional	

MFWPOPENCISPAGE Properties

Basic			
name	Text of the node.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node. You can append parameters to the URL by appending them via "&param1=value1&param2=value2"	Obligatory	
followpageswitches	If the user navigates inside the called page (e.g. switches from one page to the other) then this navigation is registered. True means: when reinvoking the page through the tree then the user come back exactly to the page where he/she stayed. False means: the user is brought back to the starting page always. For HTML pages: Registering of the navigation is only supported for HTML pages in the framebuffer. This means you need to set the framebuffersize attribute in the cisconfig.xml file correspondingly.	Obligatory	

onlyoneinstance	A page with the corresponding text is only started once inside the workplace. If the page already exists no new pages is started but the existing one is picked.	Obligatory	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
iconurl	URL for the icon in front of the text. The workplace iself is running in project "HTMLBasedGUI" - you have to go up first "../" to address your icons.	Optional	
tooltip	Text of the tooltip of the tree node.	Optional	
tooltipid	Text ID of the tooltip.	Optional	

MFWPOPENCISPOPUP Properties

Basic			
name	Text of the node.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node. You can append parameters to the URL by appending them via "&param1=value1&param2=value2"	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
iconurl	URL for the icon in front of the text. Must start with "../project".	Optional	
tooltip	Tooltip of the node.	Optional	
tooltipid	Text ID of tooltip.	Optional	
width	Set the dimension of the popup in pixels. (width)	Optional	1 2 3 int-value
height	Set the dimension of the popup in pixels. (height)	Optional	1 2

			3
			int-value
left	Set the dimension of the popup in pixels. (left)	Optional	1
			2
			3
			int-value
top	Set the dimension of the popup in pixels. (top)	Optional	1
			2
			3
			int-value

MFWPOPENCISTARGET Properties

Basic			
name	Text of the node.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node. You can append parameters to the URL by appending them via "&param1=value1&param2=value2".	Obligatory	
target	Name of the target Frame in which the CIS page is going to be opened. During workplace definition each frame you define gets assigned a target-id.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
iconurl	URL for the icon in front of the text. Must start with "../project".	Optional	
tooltip	Tooltip of the node.	Optional	
tooltipid	Text ID of the tooltip.	Optional	

MFWPCALLBACK Properties

Basic			
name	Text of the item.	Obligatory	
textid	Text ID of the items text.	Optional	
class	Command that is executed if the node is selected.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
iconurl	Icon of the node. Must be something like "../project/... - the workplace itself is running in project "HTMLBasedGUI", you have to move up first as consequence.	Optional	
tooltip	Tooltip of the item.	Optional	
tooltipid	Tooltip Text ID of the item.	Optional	

MFWPOPENHTMLPAGE Properties

Basic			
name	Text of the node.	Optional	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node.	Optional	
followpageswitches	If the user navigates inside the called page (e.g. switches from one page to the other) then this navigation is registered. True means: when reinvoking the page through the tree then the user come back exactly to the page where he/she stayed. False means: the user is brought back to the starting page always. For HTML pages: Registering of the navigation is only supported for HTML pages in the framebuffer. This means you need to set the framebuffersize attribute in the cisconfig.xml file correspondingly.	Optional	
onlyoneinstance	A page with the corresponding text is only started once inside the workplace. If the page already exists no new pages is started but the existing one is picked.	Optional	true false
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			

draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
iconurl	URL for the icon in front of the text. Must start with "../project"	Optional	
tooltip	Tooltip of the node.	Optional	
tooltipid	Text ID of the tooltip.	Optional	

MFWPOPENHTMLPOPUP Properties

Basic			
name	Text of the node.	Optional	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
iconurl	URL for the icon in front of the text. Must start with "../project"	Optional	
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
tooltip	Tooltip of the node.	Optional	
tooltipid	Text ID of the tooltip.	Optional	
width	Set the dimension of the popup in pixels. (width)	Optional	1 2 3 int-value
height	Set the dimension of the popup in pixels. (height)	Optional	1 2 3 int-value
left	Set the dimension of the popup in pixels. (left)	Optional	1 2 3 int-value

top	Set the dimension of the popup in pixels. (top)	Optional	1
			2
			3
			int-value

MFWPOPENHTMLTARGET Properties

Basic			
name	Text of the node.	Obligatory	
textid	Multi language dependent text that is displayed inside the control. The "textid" is translated into a corresponding string at runtime. Do not specify a "name" inside the control if specifying a "textid".	Optional	
activityurl	URL to be started when user clicks on node.	Obligatory	
target	Name of the target Frame in which the HTML Page is going to be opened. When defining a workplace page you assign a target-id per frame.	Obligatory	
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor's tree view.	Optional	
Appearance			
iconurl	URL for the icon in front of the text Must start with "../project".	Optional	
draginfo	Any information that is useful to react on a drop event. Characters ' and \ are not allowed.	Optional	
tooltip	Tooltip of the node.	Optional	
tooltipid	Text ID of the tooltip.	Optional	

Customizing the MFWPFUNCTIONS Behavior

The `mfworkplaceeventlistener` property of MFWPBOOTSTRAPINFO defines a Java class name. This class listens to events raised by the workplace and reacts accordingly. Examples for such events are context menu requests, or reactions to opening, closing, removing or switching of content pages. You can write your own event handler class by providing a Java class which implements the `com.softwareag.cis.workplace.IMFWorkplaceEventListener2` interface. See the Javadoc documentation (see also *Developing Java Extensions* in the *Ajax Developer* documentation).

Often, you do not want to write a complete event handler class. Instead, you would like to keep most of the default behavior, but simply customize pop-up messages and/or the shown context menus for the different nodes in the function tree. The following topics describe how to do simple customizations for the default event handler implementation.

You start with the class `com.softwareag.cis.workplace.MFCustomEventListener`. If you only want to customize pop-up messages, you can simply extend this class. If you would like to customize context menus and/or reactions to other events, you can use the `MFCustomEventListener` class as a template for writing your own custom event listener. The `MFCustomEventListener` class extends the `MFEventListenerBase` class which implements basic event reactions.

The following topics are covered below:

Customizing Pop-up Messages

If you only want to customize pop-up messages and keep the default context menu and event reaction, proceed as follows.

Create a class (for example, `MyCustomEventListener`) and implement the following methods (see also the example below):

- `String getPopupMessageNumberOfWorkplaceActivitiesReached(...)`

- `String getPopupTitelMaxNumberOfWorkplaceActivitiesReached(...)`

- `String getPopupMessagePopupMenuClosedByUser()`

- `String getPopupTitelPopupMenuClosedByUser()`

```
public class MyCustomEventListener extends MFCustomEventListener
{
protected String getPopupMessageNumberOfWorkplaceActivitiesReached(
                    int    maxactivities)
{
    return "THIS IS MY OWN MESSAGE";
}

protected String getPopupTitleNumberOfWorkplaceActivitiesReached(
                    int    maxactivities)
{
    return "THIS IS MY OWN POP-UP TITLE";
}

protected String getPopupMessagePopupMenuClosedByUser()
{
    return "THIS IS MY OWN MESSAGE";
}

protected String getPopupTitlePopupMenuClosedByUser()
{
```

```
    return "THIS IS MY OWN POP-UP TITLE";
}
}
```

Specify the `MyCustomEventListener` class in your bootstrapinfo (see below) and put the class file into the classpath of your web application.

```
<mfwpbootstrapinfo
    defaultcontentpage="/HTMLBasedGUI/empty.html"
        ...
        mfworkplaceeventlistener="com.mycompany.MyCustomEventListener"
...
```

Customizing Context Menus

If you would like to have your own context menus, you need to implement the following methods:

- `TREECollection buildContextMenu(...)`

- `TREECollection buildDropMenu(...)`

- `TREECollection buildFunctionContextMenu(...)`

- `TREECollection buildMFTopicContextMenu(...)`

All of these methods return a `TREECollection` object with the nodes for the context menu. For details of the different methods, see the corresponding Javadoc documentation of the `com.softwareag.cis.workplace.MFEventListenerBase` class. See also *Developing Java Extensions* in the *Ajax Developer* documentation.

Recommendation:

1. Write your own class (for example, `AnotherCustomEventListener`) which extends `MFEventListenerBase`.

2. Use the `MFCustomEventListener` class as a template. Here you can see how a `TREECollection` object is built. You can copy all required information and paste it in your own class.

A `TREECollection` is an object which describes a tree of nodes. Each node implements some standard commands such as **Remove**, **Cut** or **Paste**. If you look at the `MFCustomerEventListener` class, you will see the class `MFCustomMenuNodeInfo` which extends the class `MFMenuNodeInfoBase`. The `MFMenuNodeInfoBase` class contains the implementation of a set of standard commands which are defined as `CMDID_*` fields in the class. See the corresponding Javadoc documentation for details (see also *Developing Java Extensions* in the *Ajax Developer* documentation). You can reuse the standard commands, or you can implement your own commands.

Recommendation for implementing your own commands:

1. Write your own node class (for example, `MyCustomMenuNodeInfo`) which extends `MFMenuNodeInfoBase`.

2. In the same way as the `MFCustomEventListener` class builds the `TREECollection` objects from `MFCustomMenuNodeInfo` nodes, your `AnotherCustomEventListener` class will build the `TREECollection` objects from the `MyCustomMenuNodeInfo` nodes.

To use your newly implemented event listener class `AnotherCustomEventListener`, specify the `AnotherCustomEventListener` class in your bootstrapinfo (see below) and put the class file into the classpath of your web application.

```
<mfwpbootstrapinfo
    defaultcontentpage="/HTMLBasedGUI/empty.html"
        ...
      mfworkplaceeventlistener="com.mycompany.AnotherCustomEventListener"
...
```

Implementing Custom Event Reactions (Advanced)

If you also want to implement own reactions to other events, you create your own class (for example, `MyAdvancedEventListener`) which implements the interface `com.softwareag.cis.workplace.IMFWorkplaceEventListener2`. See the Javadoc documentation for details (see also *Developing Java Extensions* in the *Ajax Developer* documentation).

Your class must implement the `react*` methods of this interface:

```
public class MyAdvancedEventListener implements IMFWorkplaceEventListener2
{
   public void reactOnDrop(...){...}
   public Boolean reactOnCloseWindowRequest(...){...}
   ...
}
```

To add your `MyAdvancedEventListener` class to the bootstrapinfo, proceed in the same way as described in the previous topics.

Source Code for com.softwareag.cis.workplace.MFCustomEventListener

```
package com.softwareag.cis.workplace;

import com.softwareag.cis.server.util.TREECollection;

/**
 * This class is an example of a simple custom event listener based on the
 * <code>MFEventListenerBase</code> default implementation. The source code is
 * available in the documentation.
 * <p>
 * It shows how to simply customize pop-up messages, pop-up titles and/or context
 * menus without having to write a complete event listener.
 * <p>
 *
```

```
 * @see com.softwareag.cis.workplace.MFEventListenerBase
 *
 */
public class MFCustomEventListener extends MFEventListenerBase
{

    /**
     * Objects of this class represent a context menu item. It extends the
     * default implementation for context menu items {@link #MFMenuNodeInfoBase}.
     * This default implementation defines default items for the basic commands
     * like CUT, PASTE, REMOVE.
     * <p>
     *
     * @see com.softwareag.cis.workplace#MFMenuNodeInfoBase
     *
     */
    public class MFCustomMenuNodeInfo extends MFMenuNodeInfoBase
    {
        /**
         * Constructor
         *
         * @param eventListener the event listener
         */
        MFCustomMenuNodeInfo(MFEventListenerBase eventListener)
        {
            super(eventListener);
        }

        /* (non-Javadoc)
         * @see ↵
com.softwareag.cis.workplace.MFMenuNodeInfoBase#init(java.lang.String,
         *          java.lang.String, com.softwareag.cis.workplace.IMFWorkplace,
         *          com.softwareag.cis.server.util.TREECollection,
         *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral[],
         *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral,
         *          com.softwareag.cis.workplace.MFWorkplaceTopic)
         */
        protected void init(String text,
                            String image,
                            IMFWorkplace workplace,
                            TREECollection tree,
                            MFWorkplaceTreeNodeGeneral[] treeNodes,
                            MFWorkplaceTreeNodeGeneral treeNode2,
                            MFWorkplaceTopic topic)
        {
            super.init(text, image, workplace, tree, treeNodes, treeNode2, topic);
        }

        /* (non-Javadoc)
         * @see com.softwareag.cis.workplace.MFMenuNodeInfoBase#init(int,
         *          com.softwareag.cis.workplace.IMFWorkplace,
         *          com.softwareag.cis.server.util.TREECollection,
```

```
 *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral[],
 *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral,
 *          com.softwareag.cis.workplace.MFWorkplaceTopic)
 */
protected void init(int cmdid,
                    IMFWorkplace workplace,
                    TREECollection tree,
                    MFWorkplaceTreeNodeGeneral[] treeNodes,
                    MFWorkplaceTreeNodeGeneral treeNode2,
                    MFWorkplaceTopic topic)
{
    super.init(cmdid, workplace, tree, treeNodes, treeNode2, topic);
}


}


/*
 * (non-Javadoc)
 *
 * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#buildDropMenu(com.softwareag.cis.workplace.IMFWorkplace,
 *          com.softwareag.cis.workplace.MFWorkplaceTopic,
 *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral,
 *          com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral[])
 */
protected TREECollection buildDropMenu(IMFWorkplace workplace,
                                       MFWorkplaceTopic topic,
                                       MFWorkplaceTreeNodeGeneral targetNode,
                                       MFWorkplaceTreeNodeGeneral[] droppedItems)
{
    TREECollection menu = new TREECollection();
    MFCustomMenuNodeInfo menuNode = null;
    if (targetNode.getOpened() == 2)
    {
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_MOVEBEFORE, workplace, ↵
topic.getTree(), droppedItems, targetNode, topic);
        menu.addTopNode(menuNode, true);
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_MOVEBEHIND, workplace, ↵
topic.getTree(), droppedItems, targetNode, topic);
        menu.addTopNode(menuNode, true);
    }
    else
    {
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_MOVEASFIRST, workplace, ↵
topic.getTree(), droppedItems, targetNode, topic);
        menu.addTopNode(menuNode, true);
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_MOVEASLAST, workplace, ↵
topic.getTree(), droppedItems, targetNode, topic);
```

```
                menu.addTopNode(menuNode, true);
        }
        return menu;
    }


    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#buildContextMenu(com.softwareag.cis.workplace.IMFWorkplace,
     *      com.softwareag.cis.workplace.MFWorkplaceTopic,
     *      com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral,
     *      com.softwareag.cis.workplace.MFWorkplaceTreeNodeGeneral[])
     */
    protected TREECollection buildContextMenu(IMFWorkplace workplace,
                                              MFWorkplaceTopic topic,
                                              MFWorkplaceTreeNodeGeneral item,
                                              MFWorkplaceTreeNodeGeneral[] selection)
    {
        TREECollection tree = topic.getTree();
        TREECollection menu = new TREECollection();


        // -------------------- Show with sub menu
        MFCustomMenuNodeInfo menuNode = null;
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_SHOW, workplace, tree, selection, ↵
null, topic);
        menu.addTopNode(menuNode, false);


        MFCustomMenuNodeInfo subNode = null;
        subNode = new MFCustomMenuNodeInfo(this);
        subNode.init(MFMenuNodeInfoBase.CMDID_SHOW_CONTENT_FRAME, workplace, tree, ↵
selection, null, topic);
        menu.addSubNode(subNode, menuNode, true, false);


        subNode = new MFCustomMenuNodeInfo(this);
        subNode.init(MFMenuNodeInfoBase.CMDID_SHOW_NEW_WINDOW, workplace, tree, ↵
selection, null, topic);
        menu.addSubNode(subNode, menuNode, true, false);


        // --------------- CUT
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_CUT, workplace, tree, selection, ↵
null, topic);
        menu.addTopNode(menuNode, true);


        // -------------- PASTE
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_PASTE, workplace, tree, selection, ↵
null, topic);
        menu.addTopNode(menuNode, true);
        if (super.getClipboardSize() == 0 ||
```

```
            item.getOpened() == 2) menuNode.setInactive(true);

        // --------------- Separator
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init("&SEPARATOR", null, workplace, tree, selection, null, topic);
        menu.addTopNode(menuNode, true);

        // --------------- REMOVE
        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_REMOVE, workplace, tree, selection, ↵
null, topic);
        menu.addTopNode(menuNode, true);

        return menu;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#buildFunctionContextMenu(com.softwareag.cis.workplace.IMFWorkplace,
     *        com.softwareag.cis.workplace.MFWorkplaceTopic)
     */
    protected TREECollection buildFunctionContextMenu(IMFWorkplace workplace,
                                            MFWorkplaceTopic selectedTopic)
    {
        TREECollection menu = new TREECollection();
        MFCustomMenuNodeInfo menuNode = null;

        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_REFRESHTOPIC, workplace, ↵
selectedTopic.getTree(), null, null, selectedTopic);
        menu.addTopNode(menuNode, true);

        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_REMOVEALL, workplace, ↵
selectedTopic.getTree(), null, null, selectedTopic);
        menu.addTopNode(menuNode, true);

        return menu;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#buildMFTopicContextMenu(com.softwareag.cis.workplace.IMFWorkplace,
     *        com.softwareag.cis.workplace.MFWorkplaceTopic)
     */
    protected TREECollection buildMFTopicContextMenu(IMFWorkplace workplace,
                                            MFWorkplaceTopic selectedTopic)
    {
```

```
        TREECollection menu = new TREECollection();
        MFCustomMenuNodeInfo menuNode = null;

        menuNode = new MFCustomMenuNodeInfo(this);
        menuNode.init(MFMenuNodeInfoBase.CMDID_REFRESHTOPIC, workplace, ↵
selectedTopic.getTree(), null, null, selectedTopic);
        menu.addTopNode(menuNode, true);
        return menu;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#getMaxNumberActivitiesMode()
     */
    protected int getMaxNumberActivitiesMode()
    {
        return MAX_NUMBER_ACTIVITIES_POPUP;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#getPopupMessageNumberOfWorkplaceActivitiesReached(int)
     */
    protected String getPopupMessageNumberOfWorkplaceActivitiesReached(int ↵
maxactivities)
    {
        // use default
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#getPopupTitelMaxNumberOfWorkplaceActivitiesReached(int)
     */
    protected String getPopupTitelMaxNumberOfWorkplaceActivitiesReached(int ↵
maxactivities)
    {
        // use default
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#getPopupMessagePopupMenuClosedByUser()
```

```
     */
    protected String getPopupMessagePopupMenuClosedByUser()
    {
        // use default
        return null;
    }


    /*
     * (non-Javadoc)
     *
     * @see ↵
com.softwareag.cis.workplace.MFEventListenerBase#getPopupTitelPopupMenuClosedByUser()
     */
    protected String getPopupTitelPopupMenuClosedByUser()
    {
        // use default
        return null;
    }

}
```

Session Management inside the Workplace

When the user selects functions in the MFWPFUNCTIONS frame, then pages are opened in the content frame, or as pop-ups or in a named target frame.

The workplace offers a "multi document interface" - i.e. you can work in parallel in several activities and you can switch between these activities. This structure is reflected in the server-side session structure. The section *Details on Session Management* in the *Special Development Topics* explains this in a detailed way. However, some information is given below.

The session management of Application Designer knows sessions (typically representing a browser instance) and subsessions (reflecting a user activity with a defined life cycle). A session contains one or more subsessions. Inside one subsession, the adapter object are kept which are required by a page or a page sequence. Subsessions are isolated from one another.

The workplace proceeds in the following way:

■ Every activity that is started inside the content is represented by a subsession of its own. If you have opened five Application Designer pages via the function tree inside the content area of the workplace, then there are five subsessions on the server side. If the user navigates between the activities (e.g. via the MFWPACTIVEFUNCTIONS frame), then from session point of view the user navigated between subsessions.

■ The workplace itself also occupies one subsession. If Application Designer pages are opened in a pop-up or in a named target, then these pages are living inside the subsession of the workplace.

When programming content pages, you do not notice the session management: every page that you design and test in the Layout Painter behaves in the same way in the workplace. Due to the separation into subsessions, you are not aware of "neighboring" subsessions.

Other Frames

You can add any further frames to the multi frame page of the workplace, as described in the section *Multi Frame Pages*. The workplace is just a functional framework using this technology - but not limiting it somehow.

Example: in the demo workplace, you see a NEWS frame below the MFWPFUNCTIONS frame that holds a certain HTML document.

Via the node types `MFWorkplaceTreeNodeCISTarget` and `MFWorkplaceTreeNodeHTMLTarget`, you can directly load pages into given frames, but you can also use the frames from your normal applications.

Workplace API for Dynamic Manipulation

Internally, the workplace is started when the workplace frameset page is loaded. So far you got to know the framework to set up the workplace in a dynamic way by implementing the bootstrap class referenced in the MFWPFUNCTIONS frame. "In a dynamic way" means that there is a program to provide for the required data - the program can build the function trees on its own, e.g. based on the user's role.

But you can also dynamically manipulate the workplace. There are two typical usages:

- You can exchange all workplace definitions dynamically. Maybe you offer the user a "reduced" workplace just allowing the user to log on at the beginning. Afterwards, the "real" workplace for the user is built up - containing all functions available for the user.

- You can manipulate workplace definitions in an existing workplace. For example, you modify the title of an activity that is shown in the MFWPACTIVEFUNCTIONS area. Or you want to add certain nodes to an existing tree.

For this purpose, there is a Java API containing the workplace functions that you can use from your adapter code.

Interface IMFWorkplace

The interface `IMFWorkplace` contains the methods you can call. The interface is accessible inside an adapter through the session context in the following way:

```
IMFWorkplace wp = (IMFWorkplace)findSessionContext().
                  lookup(IMFWorkplace.IWORKPLACE_LOOKUP,false);
```

Pay attention: the interface instance is only returned if the page is running inside the workplace. If a page is running, for example, inside the Layout Painter or if a page is directly started via the "StartCISPage" servlet, then "null" will returned.

The `IMFWorkplace` interface contains a set of methods for accessing and manipulating the workplace. There is one method `updateWorkplace(...)` that is especially important: when changing the workplace you have to call this method at the end to make the changes visible in the user interface. The method expects an adapter to be passed: this is the adapter that currently processes the request from the browser.

Exchanging complete MFWorkplaceInfo

Via the method `exchangeMFWorkplaceInfo(...)`, you can exchange the complete settings of the workplace. Example: you may have a logon screen in which the adapter method for handling the logon looks as follows:

```
public void onLogon()
{
    // check user and password
    ...
    ...
    // build up workplace for user
    MFWorkplaceInfo wi = new MFWorkplaceInfo();
    ...
    ...
    ...
    // exchange workplace
    IMFWorkplace wp;
    wp = ↵
(IMFWorkplace)findSessionContext,lookup(IMFWorkplace.IWORKPLACE_LOOKUP,false);
    wp.exchangeMFWorkplaceInfo(wi);
    wp.updateWorkplace(this);
}
```

Opening Pages in the Workplace

There are the functions that you can use to open new pages in the content area:

- `showPageInWorkplace`

- `addPageToWorkplace`

- `showHTMLPageInWorkplace`

- `addHTMLPageToWorkplace`

You either open Application Designer pages (`...Page...`) or URLs (`...HTML...`). Pages are either added as new activities (`add...`) or the workplace first finds out whether a page with the same name was already started before opening a new one (`show...`).

There is the method with which you can switch to an already opened activity inside the workplace:

- `switchToSubsession`

Fine Granular Updates

There is a method that you use in order to update the title that is shown for the page in the MFWPACTIVEFUNCTIONS frame:

- `updatePageTitle`

There is a method that passes back the currently active `MFWorkplaceInfo` object:

- `getMFWorkplaceInfo`

Inside the `MFWorkplaceInfo` object, there are various methods for updating the object.

Example - Double Line Menu Workplace

With the available framework components

- multi frame pages,
- workplace frame controls, and
- workplace API,

you can build your own powerful workplaces that do not look like the "typical" Application Designer workplaces. Have a look at the following workplace:
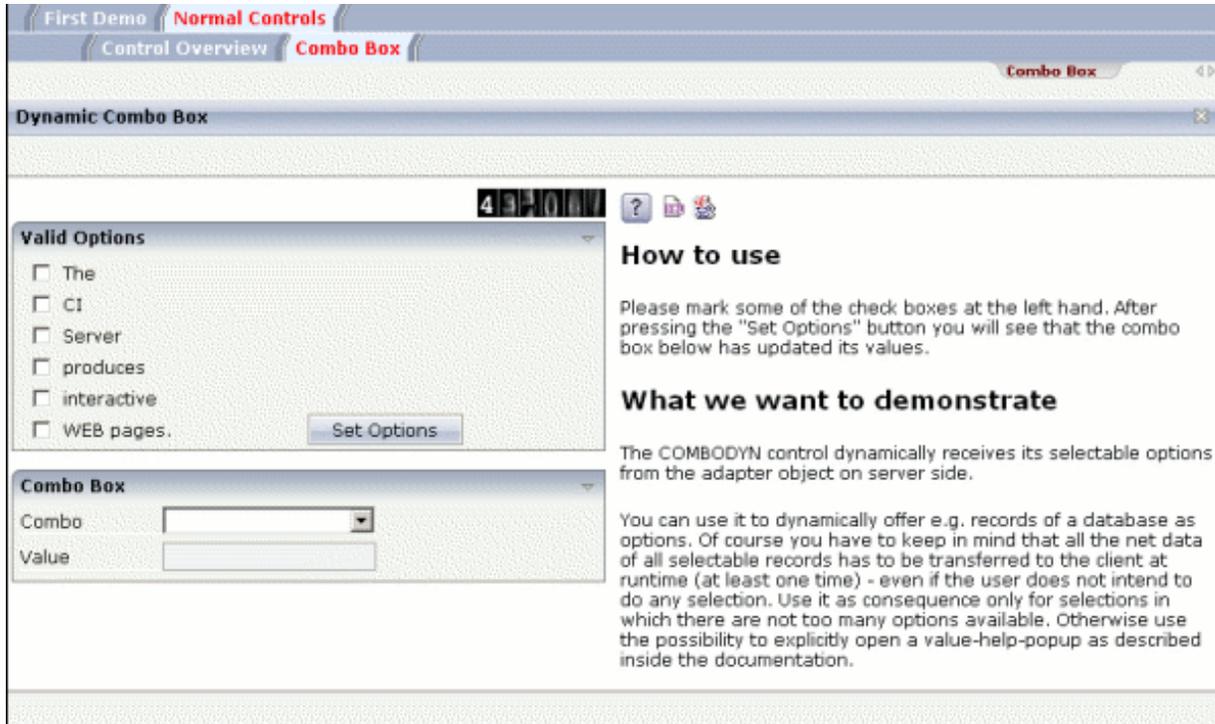
In the workplace, a small set of functions is arranged in a double line menu. When selecting the functions from the menu, the content is shown in the content frame.

The workplace's multi frame page is defined in the following way:

```
<mfpage separation="rows" sizing="*" border="0">
    <mfframeset target="AAA" separation="rows" sizing="0,41,25,*" border="0">
        <mfwpfunctions bootstrapclass="com.softwareag.cis.test25.DLWPInit">
        </mfwpfunctions>
        <mfcisframe target="DLMENU" cisurl="/cisdemos/25_dlworkplacemenu.html">
        </mfcisframe>
        <mfhtmlframe target="CURRENTACTIVITIES" ↵
url="../HTMLBasedGUI/workplace/loading.html">
        </mfhtmlframe>
        <mfhtmlframe target="CONTENT" url="../HTMLBasedGUI/workplace/loading.html">
        </mfhtmlframe>
    </mfframeset>
</mfpage>
```

The workplace holds the three workplace frames you know from a previous section: the **MFWP-FUNCTIONS** frame, though it is sized to be invisible ("0"). The bootstrap class that is referenced (*com.softwareag.cis.test25.DLWPInit*) is only a dummy and returns an empty `MFWorkplaceInfo` object.

There is a frame, DLMENU, in which by using a normal Application Designer page (*/cis-demos/25_dlworkplacmenu.html*), the double line menu is displayed. The implementation of this page on the server side looks like:

```
package com.softwareag.cis.test25;

// 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.workplace.IWorkplace;

public class DLWPMenuAdapter
    extends Adapter
{
    // ------------------------------------------------------------------------
    // property access
    // ------------------------------------------------------------------------

    public class MyDLMenuSubItem extends DLMenuSubItem
    {
        String m_url;
        public MyDLMenuSubItem(DLMenuTopItem topItem,
                               String text,
                               String url)
        {
            super(topItem, text);
            m_url = url;
        }

        public void invoke()
        {
            showPage(m_url,getText());
        }
    }

    DLMenu m_dlmenu = new DLMenu();
    public DLMenu getDlmenu() { return m_dlmenu; }

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

    public void init()
    {
        // fill menu
        DLMenuTopItem top;

        top = new DLMenuTopItem(m_dlmenu,"First Demo");
        new MyDLMenuSubItem(top,"Hello world","/cisdemos/DEMO_HelloWorld.html");  ↵


        top = new DLMenuTopItem(m_dlmenu,"Normal Controls");
        new MyDLMenuSubItem(top,"Control ↵
```

```
Overview","/cisdemos/DEMO_ControlOverview.html");
      new MyDLMenuSubItem(top,"Combo Box","/cisdemos/DEMO_ComboDyn.html");
   }

   public void showPage(String url,
                        String text)
   {
      IWorkplace wp = (IWorkplace)findSessionContext().
                        lookup(IWorkplace.IWORKPLACE_LOOKUP,false);
      if (wp != null)
      {
          wp.showPageInWorkplace(url,text);
          wp.updateWorkplace(this);
      }
   }

}
```

The class uses the workplace API for opening pages in order to make the right page visible in the content area when the user clicks into the double line menu.

Usage Example - Calling the Application Designer Workplace with Directly Opening a Page

Let us imagine the following scenario: you want to open an Application Designer workplace from somewhere else (e.g. from a portal application), showing your workplace setup just as normal. In the workplace, you want one (or more) application(s) to be already opened.

To do so, you have to:

■ define one starter page that you call from the "somewhere else" application,

■ pass the name of the HTML page to be opened inside the workplace as a parameter to this starter page; the adapter of the starter page will write this parameter into the session context and will then execute a "switch page" to the workplace,

■ define an empty page inside the workplace that looks at the session context and uses the workplace API functions to start the application inside the workplace.

Step by step:
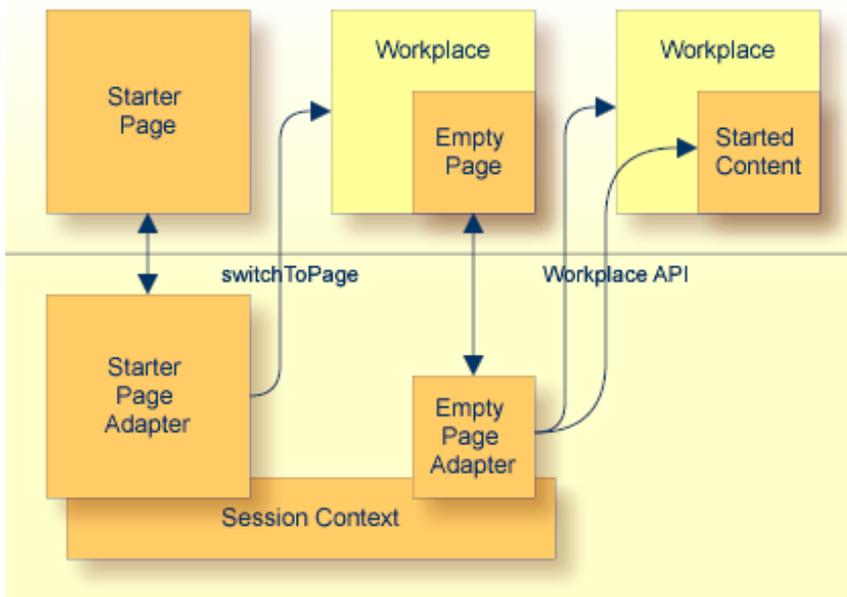
The name of the starter page in this example is */cisworkplace/starter_withStartPage.html*. Its XML code is quite simple:

```
<page model="StarterWithStartPageAdapter">
    <pagebody>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

The adapter code is:

```
import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class StarterWithStartPageAdapter
    extends Adapter
{
    String m_pageToBeStarted;
    public void setPageToBeStarted(String value)
    {
        m_pageToBeStarted = value;
    }

    public void reactOnDataTransferEnd()
    {
        super.reactOnDataTransferEnd();
        if (m_pageToBeStarted != null)
```

```
        {
            // prepare empty for starting
            findSessionContext().bind("test/pageToBeStarted",m_pageToBeStarted);
            // start workplace
            switchToPage("workplace_withStartPage.html");
        }
        else
        {
            outputMessage(MT_ERROR,"No page found to be started!");
        }
    }

}
```

As you can see from the code, the starter page's adapter does nothing else than providing the property `pageToBeStarted` and a method `reactOnDataTransferEnd` that is processed at the end of the set phase. In the method, the `pageToBeStarted` property is written into the session context and a switch to the workplace page *workplace_withStartPage.html* is done.

This starter page is opened in the following way:

```
http://<host>:<port>/<webapp>/servlet/StartCISPage?PAGEURL=/cisworkplace/starter_withStartPage.html&pageToBeStarted=/<project>/<page.html>
```

The starter page itself will only become visible if an error occurs (e.g. no parameter `pageToBeStarted` is passed). Otherwise, it will always switch to the workplace page.

The workplace page is just a normal Application Designer workplace that you build using Application Designer's Layout Painter:

```
<mfpage separation="rows" sizing="40,20,*">
    <mfhtmlframe target="TITLEPAGE" url="../cisworkplace/workplace/header.html" ↵
resizable="false"
     withborder="false" scrolling="false" framestyle="border: 0px solid #000000" ↵
marginheight="0" marginwidth="0">
    </mfhtmlframe>
    <mfwpactivefunctions resizable="false" withborder="false" scrolling="false"
     framestyle="border: 0px solid #000000">
    </mfwpactivefunctions>
    <mfframeset target="ZZZ" separation="cols" sizing="265,*">
        <mfframeset target="LEFTPART" separation="rows" sizing="*,87">
            <mfwpfunctions bootstrapclass="WorkplaceWithStartPageProvider"
            serversidescrolling="false" framestyle="border: 1 solid #808080;">
            </mfwpfunctions>
          <mfhtmlframe target="NEWS" url="../cisworkplace/workplace/welcome.html" ↵
resizable="true"
             withborder="false" scrolling="true" framestyle="border: 1px solid ↵
#808080">
            </mfhtmlframe>
        </mfframeset>
        <mfwpcontent resizable="true" withborder="true" scrolling="false" ↵
framestyle="border: 1 solid #808080;">
```

```
        </mfwpcontent>
    </mfframeset>
</mfpage>
```

It somewhere contains the **MFWPFUNCTIONS** frame that internally points to a class, called "bootstrap" class. This is the class that (as runtime object) configured the workplace with its topics and function trees:

```
public class WorkplaceWithStartPageProvider
    implements IMFWorkplaceBootstrapInfoProvider2
{
    public MFWorkplaceInfo getWorkplaceInfo()
    {
        // create workplace info object, define the page that is shown
        // in content area if no other content page is shown
        MFWorkplaceInfo result = new ↵
MFWorkplaceInfo("/cisworkplace/empty_withStartPage.html");

        MFWorkplaceTopic topic;
        TREECollection tree;
        MFWorkplaceTreeNodeFolder folder;
        MFWorkplaceTreeNodeCISPage page;

        // create first topic
        topic = new MFWorkplaceTopic("Topic 1",result);
        tree = topic.getTree();
        folder = new MFWorkplaceTreeNodeFolder("Simple Demos");
        tree.addTopNode(folder,false);
        page = new MFWorkplaceTreeNodeCISPage("Hello ↵
World","/cisdemos/DEMO_HelloWorld.html",true,true);
        tree.addSubNode(page,folder,true,false);
        ...
        ...
        ...
```

This is a "just normal" bootstrap class implementation, opening the page /cisworkplace/empty_with-StartPage.html as an empty page. Remember: the empty page is the one that is shown inside the workplace content when no other application is opened. It is shown as the default content page inside the workplace with no active function.

Now let us have a look at the empty page. The XML code is again very simple (typically the empty page is some kind of background page that, for example, contains some nice images).

```
<page model="EmptyWithStarterAdapter">
    <pagebody>
    </pagebody>
</page>
```

The important thing is what happens inside the adapter of the empty page:

```
import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;
import com.softwareag.cis.workplace.IWorkplace;

public class EmptyWithStarterAdapter
    extends Adapter
{

    boolean m_firstCall = true;

    public void reactOnDataTransferEnd()
    {
        super.reactOnDataTransferEnd();
        // call workplace
        if (m_firstCall == true)
        {
            String pageToBeStarted =
                (String)findSessionContext().lookup("test/pageToBeStarted",false);
            if (pageToBeStarted != null)
            {
                IWorkplace wp =
                    ↵
(IWorkplace)findSessionContext().lookup(IWorkplace.IWORKPLACE_LOOKUP,false);
                if (wp != null)
                {
                    wp.addPageToWorkplace(pageToBeStarted,"Page to be started");
                    wp.updateWorkplace(this);
                }
            }
            m_firstCall = false;
        }
    }
}
```

The first time the adapter is called (i.e. the first time the page is shown), it checks if someone left an information inside the session context to start a certain page - exactly the information that is written into the context in the starter page. If there is some information, the corresponding page is opened as the content page of the workplace.

18 Integration into Other Workplace/Portal Scenarios

In many cases, you want to run Application Designer pages inside your own environments that are outside of Application Designer.

The requirements of other workplace/portal environments are:

■ Pages must be accessible by URLs.

■ There must be a possibility to pass information to pages. For example, user management is provided by a portal management. The result (the name of the user who is currently logged in) should be passed to applications for further processing.

To call Application Designer pages with a URL: normally each single Application Designer page can be called individually with a corresponding URL. "Normally" means that this is true from the Application Designer perspective - maybe it is not completely true from your application's perspective: one page requires a certain page to be run first, etc.

To call an Application Designer page, simply use the following URL:

```
http://<host>:<port>/cis/StartCISPage?PAGEURL=<pageURL>
```

Replace the `<pageURL>` with the URL of the wanted Application Designer page and it will be opened.

For information on additional parameters that you can pass via the StartCISPage servlet, see *Appendix E - StartCISPage Servlet* .

Passing Parameters to your Application Designer Page

You can append any number of parameters to the URL mentioned in the previous section. Each parameter consists of the sequence "&`<paramName>`=`<paramValue>`". If you want to pass the `customerId` to a "customer detail" page, the URL would look like:

```
http://<host>:<port>/cis/StartCISPage?PAGEURL=/appxyz/customerdetail.html&customerId=4711
```

Each parameter is bound to a corresponding property of the page adapter. For example, the "customer detail" page is hooked on the adapter `CustomerDetailAdapter`. Therefore, it must provide a corresponding `customerId` property to which the parameter is passed at runtime:

```
public class CustomerDetailAdapter extends Adapter
{
    ...
    public void setCustomerId(String value)
    {
        ...
    }
    ...
}
```

19 Extended Functions in the Application Designer Workplace

The **previous** section covered the "normal usage mode" of the Application Designer workplace. But there are extended functions allowing you to more interactively operate with the Application Designer workplace.

These functions include:

■ Drag-and-drop interface: you can drag and drop icons within the hierarchy of the workplace. You can drop information that was dragged from any content page into the workplace.

■ Right mouse button interface on workplace nodes.

These functions allow the users to arrange their workplace settings (e.g. functions that are part of their workplace) in a simple way on their own.

Interface IMFWorkplaceEventListener

The base of the extended functions is the interface `com.softwareag.cis.workplace.IMFWorkplaceEventListener`. The interface contains methods that are called on certain events. The most important methods are:

■ `reactOnContextMenuRequest`
This method is called when a user presses the right mouse button on a tree node of the workplace. Your implementation can build up a context menu - just as normal context menus are built up inside the tree management.

■ `reactOnDrop`
This method is called when the user performes a drag-and-drop operation inside the workplace. Your implementation may copy the dragged nodes below the dropped node or may open a pop-up menu in which the user is asked about what to do with the dragged items.

■ `reactOnDropGeneric`
This method is called when the user performes a drag-and-drop operation from any DROPICON control that is part of content pages.

The implementation of the interface is completely "yours". Use the workplace interface you got to know in the previous section to manipulate the workplace, e.g. to access the currently shown tree and to manipulate it.

An instance of the workplace event handler is passed by calling the method `registerMFWorkplaceEventListener` inside the `MFWorkplaceInfo` class:

```
MFWorkplaceInfo workplaceInfo = new MFWorkplaceInfo("/HTMLBasedGUI/empty.html",
                                    "../softwareag/styles/CIS_DEFAULT.css"); ↵

workplaceInfo.setSynchTabNavigation(true);
workplaceInfo.registerMFWorkplaceEventListener(new XYZ(...));        ↵
```

See the Java API documentation for detailed information.

Example

Among other features, the Application Designer demo workplace framework provides the following:

■ Right mouse button click on a workplace menu item (copy, cut, paste, etc.).

■ Drag-and-drop within the workplace menu (to move menu items).

Have a look at the event listener source coding. You can find it in your installation at:

<installdir>/cis/cisdemos/src/com/softwareag/cis/demoworkplace/CISDemoWorkplaceEventListener

20 Building Own Workplaces as a Frameset Definition

A set of functions is available which simplify the usage of Application Designer HTML pages inside a given HTML frameset definition. The functions are not only usable in the scope of workplace/portal management, but can also be used apart from this.

Basics

The basic functions cover the following aspects:

■ You can define an HTML page containing any kind of frameset you want. In this page, you design the frames, their sizes, their scroll behavior, their behavior when resizing the screen, etc. For each frame which which you want to interact, you define an identifier name.

■ You open Application Designer pages inside the frames. There are two possibilities:

1. Open these pages with a URL as described in the previous section.

2. Open these pages with adapter methods (server-side processing).

This section will focus on the second possibility since the first is just a certain usage of what is described in the previous section. This offers you an explicit control about what happens inside the frames: e.g. a page within frame "A" should be replaced by another page. Before proceeding, the user should be asked whether to store unsaved data (or not).

It is possible to communicate with frames on the client side. This means, you can build up interaction (e.g. you want to update another frame's content) without any flickering in the target frame.

Defining the Frameset

In the following screen, a page is shown which is divided into three frames:

The corresponding frameset definition of the page is:

```
<html>

<head>
<title>New Page 2</title>
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
</head>

  <frameset cols="200,*">
    <frameset rows="*,*">
      <frame name="lefttop" ↵
src="/cis/servlet/StartCISPage?PAGEURL=/cisdemos/frameleft.html">
      <frame name="leftbottom" src="blank.html">
    </frameset>
    <frame name="right" src="blank.html">
  </frameset>

</html>
```

The frameset contains three frames with the IDs `lefttop`, `leftbottom` and `right`. The `lefttop` frame opens the Application Designer page */cisdemos/frameleft.html*. This page contains buttons for some functions and acts like a "menu page".

Simple Way of Opening Pages in Frames

When choosing the **Customer Orders** button, the corresponding Application Designer page is opened in the `leftbottom` frame:
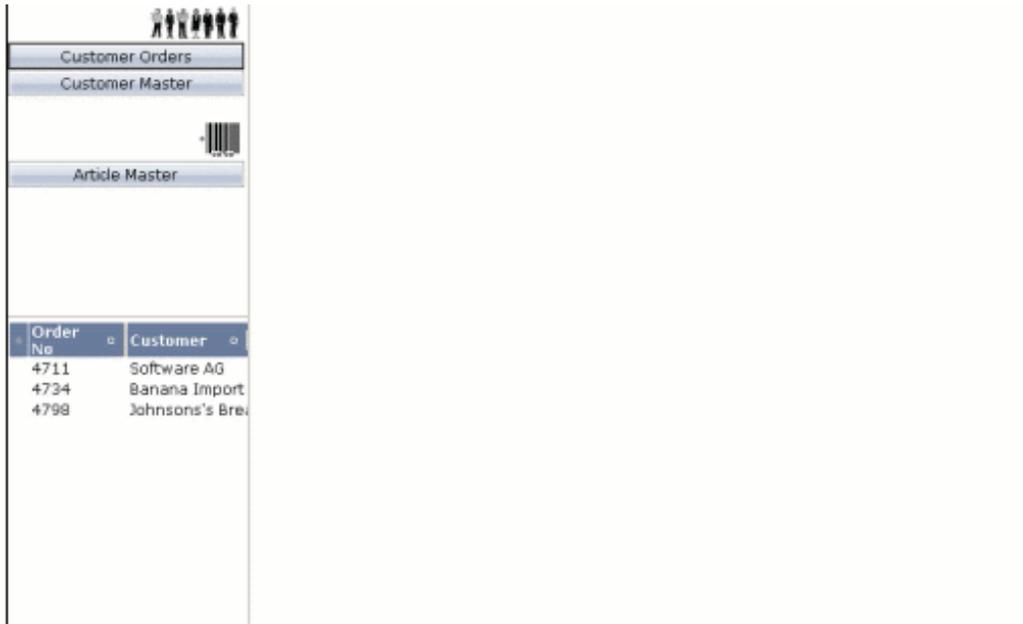
The page shows a list of customer orders. It is a normal Application Designer page. How can it be opened by choosing the **Customer Orders** button?

The /*cisdemos/frameleft.html* page (acting as a "menu page") is hooked on to a Java adapter class which looks as follows:

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

// This class is a generated one.

public class FrameLeftAdapter
    extends Adapter
{
    /** */
    public void onArticleMaster()
    {
        // TODO Auto-generated method stub
    }

    /** */
    public void onCustomerMaster()
    {
        // TODO Auto-generated method stub
```

```
    }

    /** */
    public void onCustomerOrders()
    {
        this.openCISPageInTarget("OpenCustomerOrders.html", "leftbottom");
    }
}
```

By choosing the **Customer Orders** button, the method `onCustomerOrders` is called. This method performs a method `openCISPageInTarget` inherited from class `Adapter`. The first parameter of the method is the page that is to be opened; the second parameter defines the ID of the frame in which the page is to be opened.

The page *OpenCustomerOrders.html*, which is opened when choosing the **Customer Orders** button, is running inside the same subsession as the page from which it was called. If you need to access the page adapter before opening the page inside the "leftbottom" frame, use the `findAdapter` method inside your adapter.

A More Complex Way of Opening Pages in Frames

When selecting an order in the `leftbottom` area of the previous example, a customer order page is displayed in the right frame:

The data from the order you selected is transferred into the corresponding fields of the customer order page. Have a closer look at the details.

This is the source of the adapter for listing customer orders:

```
import java.util.Iterator;

import com.softwareag.cis.server.Adapter;
import com.softwareag.cis.server.IInteractionSessionMgr;
import com.softwareag.cis.server.InteractionSessionMgrFactory;
import com.softwareag.cis.server.util.SelectableLine;
import com.softwareag.cis.server.util.TEXTGRIDCollection;
import com.softwareag.cis.util.CDate;

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

    public class Order
        extends SelectableLine
    {
        public Order(String number, String date, String customer)
        {
            m_customer = customer;
            m_date = new CDate(date);
            m_number = number;
        }

        // property >orders[*].customer<
        String m_customer;
        public String getCustomer() { return m_customer; }
        public void setCustomer(String value) { m_customer = value; }

        // property >orders[*].date<
        CDate m_date;
        public CDate getDate() { return m_date; }
        public void setDate(CDate value) { m_date = value; }

        // property >orders[*].number<
        String m_number;
        public String getNumber() { return m_number; }
        public void setNumber(String value) { m_number = value; }
    }

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

    // property >orders<
    TEXTGRIDCollection m_orders = new TEXTGRIDCollection();
    public TEXTGRIDCollection getOrders() { return m_orders; }
```

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

public void onOrderSelect()
{
    // find the selected item
    Order selectedOrder = null;
    Iterator iter = m_orders.iterator();
    while (iter.hasNext())
    {
        selectedOrder = (Order)iter.next();
        if (selectedOrder.getSelected() == true)
            break;
        else
            selectedOrder = null;
    }
    if (selectedOrder == null)
        return;
    // session management: "refresh" subsession
    String sessionId = this.m_interactionProcess.getSessionId();
    IInteractionSessionMgr iism = ↵
InteractionSessionMgrFactory.getInteractionSessionMgr();
    iism.removeSubsession(sessionId,"subsession_right");
    iism.createNewSubsession(sessionId,"subsession_right");
    // prefetch and manipulate adapter inside the refreshed subsession
    CustomerOrderDetailAdapter coda = ↵
(CustomerOrderDetailAdapter)iism.findAdapterInSubsession
        (sessionId, // sessionID
         "subsession_right", // subsessionId
         CustomerOrderDetailAdapter.class.getName(), // class
         "", // pageId, typically ""
         findPageApplication()); // application project
    coda.setNumber(selectedOrder.getNumber());
    coda.setName(selectedOrder.getCustomer());
    // navigate to page
    openCISPageInTarget("CustomerOrderDetail.html","subsession_right","right");
}

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

// property >messageType< implemented in Adapter
// property >messageShortText< implemented in Adapter
// property >messageLongText< implemented in Adapter

/** initialisation - called when creating this instance*/
public void init()
{
    m_orders.add(new Order("4711","20020706","Software AG"));
```

```
        m_orders.add(new Order("4734","20020702","Banana Import Export Ltd."));
        m_orders.add(new Order("4798","20020604","Johnsons's Bread"));
    }
}
```

With method `onOrderSelect`, the selected line is determined first.

In the next steps, frame communication is prepared and finally done. The difference to the previous "simple" scenario is that the page which is opened runs in a different subsession inside the session management of Application Designer.

Remember that each browser instance internally requests one session, and that each session is divided into various subsessions. Adapters are running inside subsessions. The subsession is responsible for keeping and releasing resources. It corresponds to one interaction process which has a defined life cycle - e.g. the data input of a customer order. For more information, see the section *Session Management*. Each subsession has an identifier - in this example, the name of the subsession is `subsession_right`. You can also create a unique ID with the class `com.softwareag.cis.util.UniqueIdMgmt`.

Our example program first removes the subsession `subsession_right`. Everything which is currently managed inside the subsession will be released. Since there is no subsession when being called the first time, no error will occur.

After releasing this subsession, a new subsession is immediately created. With the interaction session manager, you can access a method which passes back an adapter instance inside a given subsession. Like the method `findAdapter` of class `Adapter`, this method returns an adapter object which is managed inside the same subsession in which the adapter is running. With the interaction session manager, you can also access adapters inside different subsessions.

The returned adapter instance gets the selected data. Finally, the frame communication takes place: pay attention that the ID of the subsession has to be passed inside the `openCISPageInTarget` method.

When to Use the Complex Way

The complex way should be your "standard thinking" in this scenario. When dealing with Application Designer pages inside different frames, you have to take care about how you manage your sessions at the server side.

The content which runs inside the frames (e.g. the Customer Order screen) is not aware of these session management dependencies. But the designer of the workplace has to take care of the interaction possibilities inside the workplace.

Opening Normal HTML Pages inside Frames

In addition to the methods `openCISPageInTarget`, there is an equivalent method `openPageInTarget` which you inherit from the `Adapter` class. This page opens a normal HTML page inside one frame.

Frame Communication

When working with frames, it is possible to open a client-side communication channel between frames: by the client, you call an adapter method of an Application Designer page which is opened in a different frame. This is done by the method `invokeMethodInTarget(methodName,targetName)` which you inherit from the `com.softwareag.cis.server.Adapter` class.
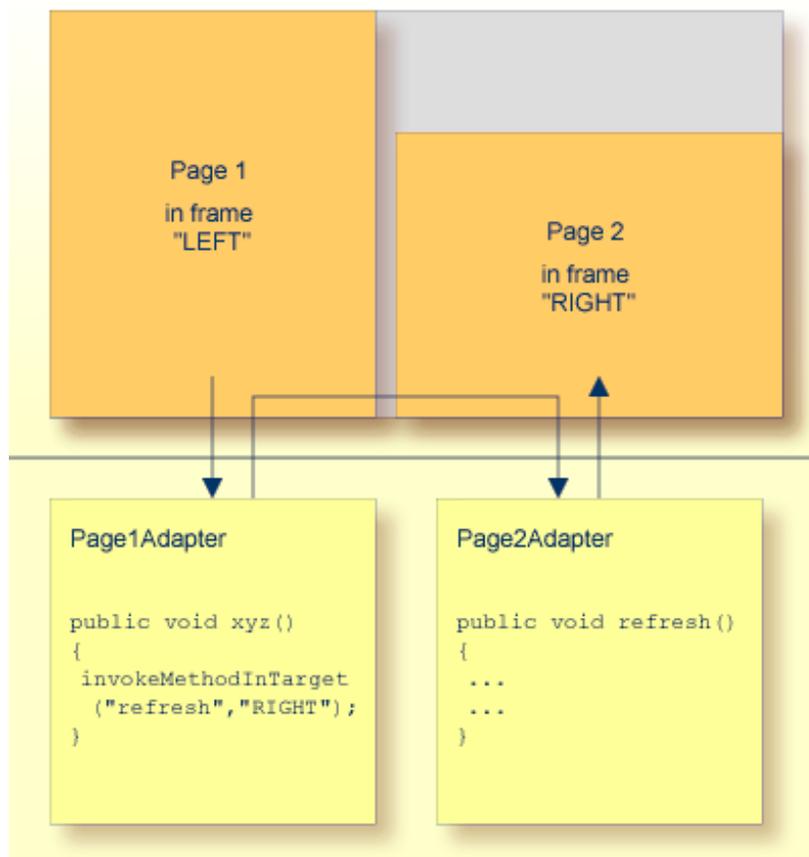
This sounds strange at first: in the adapter processing of frame `LEFT`, you call an adapter method of frame `RIGHT` and the call is executed by the client - what is the reason for this? The advantage of calling the method by the client is that the call is initiated by the page which runs inside the target frame. This means that the target frame sends the request for method execution and updates

its content as a reaction of the request's response. In other words: you can update pages in other frames without redrawing the page, i.e. without flickering.

This is a very powerful way of allowing communication between frames - but there are some restrictions which you have to keep in mind:

- The frame which initiates the interaction as well as the target frame must be in the same frameset page - in other words: they must share the same "document parent".

- The page shown in the target frame must be received by the same server and port as the page which initiates the communication. Otherwise, you will receive a JavaScript security exception - it is (by default) not allowed to establish a client communication between pages coming from different hosts.

The frame communication framework acts upon error situations in a quite tolerant way:

- If you invoke a method inside a frame target and the frame does not exist, nothing will be done on the client side.

- If you invoke a method inside a frame target and there is no valid Application Designer page inside the frame, nothing will be done.

- If you invoke a method inside a frame target and the corresponding "target adapter" does not provide the requested method, the content of the frame target's page will be synchronized with its adapter. This is similar to defining a BUTTON control and specifying a method which does not exist inside the adapter.

> **Tip:** Only use frame communication if you want to update the content of another frame page. Do not use this method for "normal" interaction between adapters, without any changes on the page.

Multiple Frame Operations

You can call the methods `openCISPageInTarget`, `openPageInTarget` and `invokeMethodInTarget` multiple times inside one request, for example, if there are multiple frames you want to manipulate at the same time.

When Building your Own Workplaces

As you learned in this section, Application Designer provides powerful mechanisms to build flexible workplace/portal scenarios. Be aware that workplace management means more than bringing up some pages into different frames. Workplace management also means, for example, that you take care of opening and closing the session state (in Application Designer: subsession state) and that you have to provide global data (like the currently logged in user) shared by all adapters, etc.