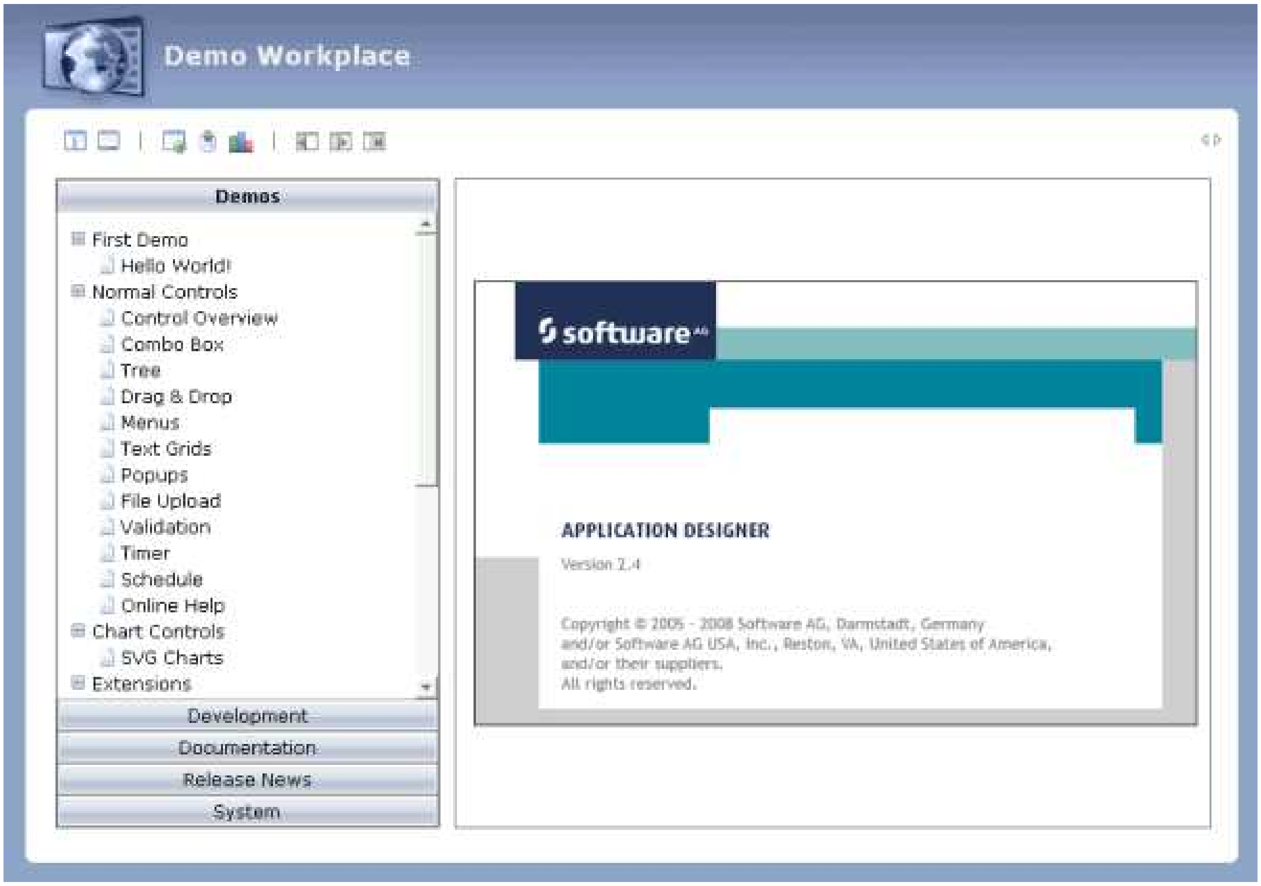
Application Designer Workplace Framework

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:

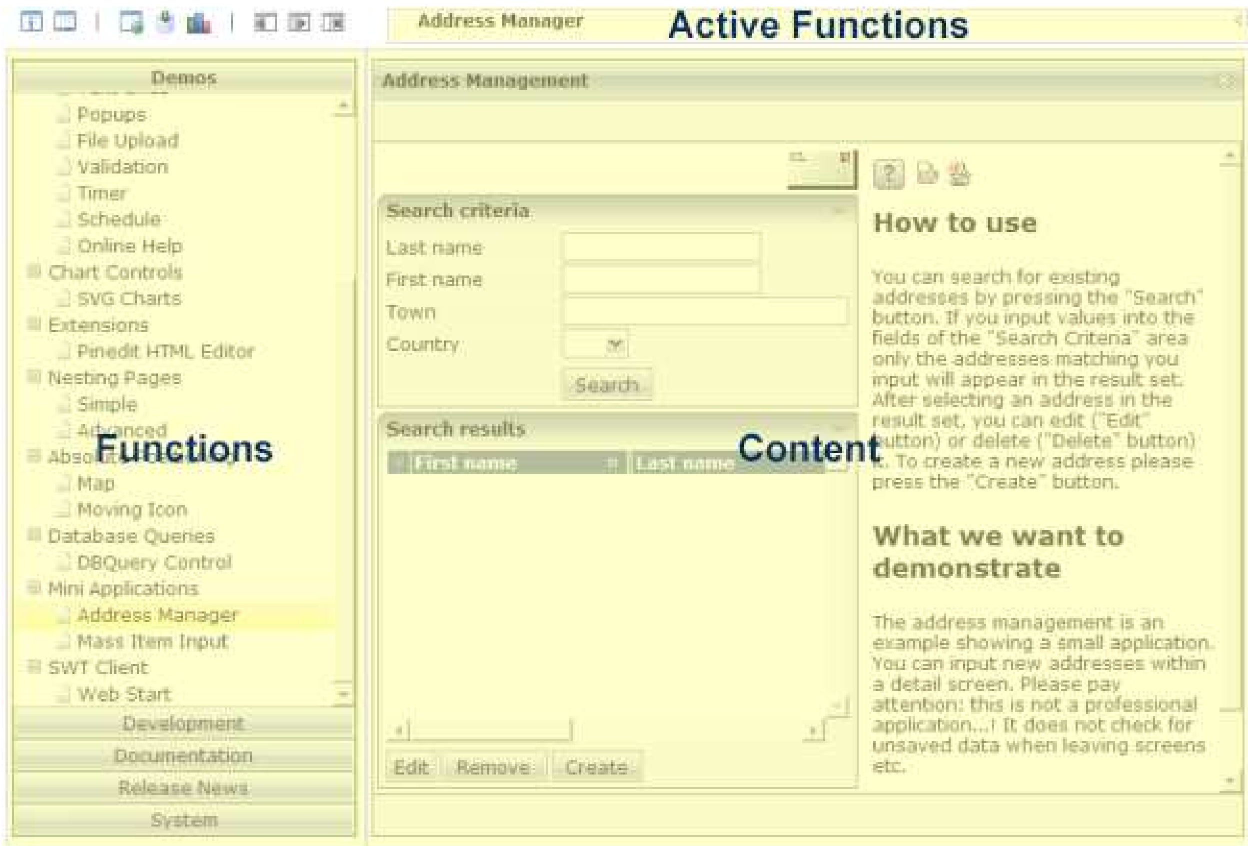
- flexible arrangement of frames,
- predefined frames containing workplace logic,
- dynamic loading of available functions,
- possibility to change the environment at runtime via the Java API,
- execution of multiple tasks between which the user can switch ("multi document interface").

This chapter covers the following topics:

- Framework Overview
- Functions Frame: MFWPFUNCTIONS

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! Use /project/directory/doc.xml as syntax, e.g. /HTMLBasedGUI/workplace/bootstrapworkplaceinfo.xml.	Optional	
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

"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.
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: MFWPBOOTSTRAPINFO

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
```

Each of the 8 sublevel tags can contain all 8 sublevel tags as subnodes, 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 (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).	Optional	
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

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 "andamp;param1=value1andamp;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 id brought back to the starting page always.	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 "andamp;param1=value1andamp;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 "andamp;param1=value1andamp;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	
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).

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
- Customizing Context Menus
- Implementing Custom Event Reactions (Advanced)
- Source Code for com.softwareag.cis.workplace.MFCustomEventListener

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.

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. 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.

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 MFWPFUNCTIONS 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 (*/cisdemos/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.

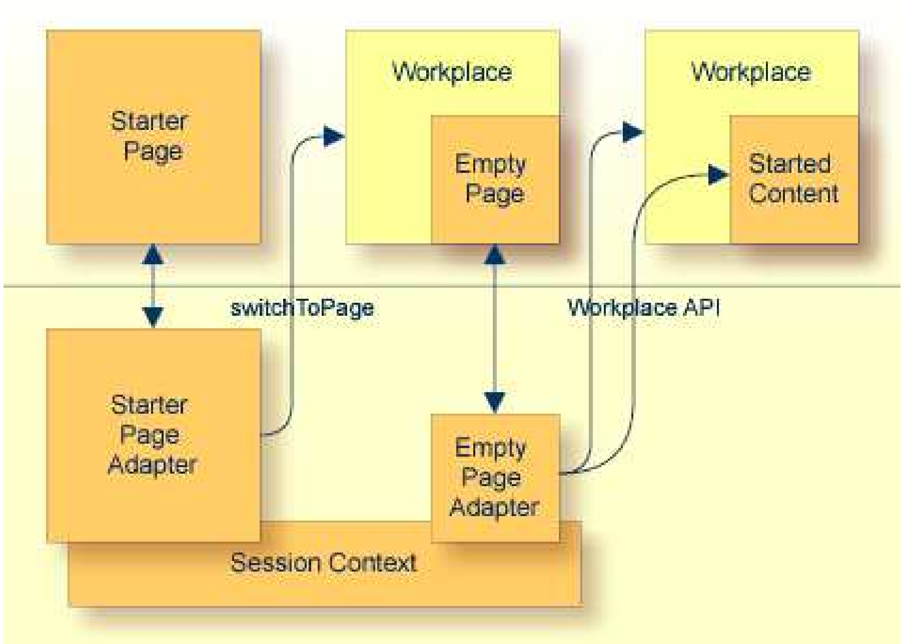
Usage Example - Calling the Application Designer Workplace with Directly Opening a Page

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_withStartPage.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.