Master Thesis

XSLGui
a graphical template editor for web content management

Author(s):
Büchler, Patrick

Publication Date:
2003

Permanent Link:
https://doi.org/10.3929/ethz-a-004916864

Rights / License:
In Copyright - Non-Commercial Use Permitted
Diploma Thesis

XSLGui

A Graphical Template Editor for Web Content Management

Patrick Büchler, D-INFK
buezi@gmx.ch

February 28th 2003

Institute for Information Systems
Swiss Federal Institute of Technology (ETH Zürich)

Diploma Professor:
Prof. Moira C. Norrie

Supervisor:
Michael Grossniklaus
Abstract

The XCM Sitemanager is a graphical tool, which greatly simplifies the construction of large websites with the CMServer content management system, developed at the Globis group at the ETH Zurich. However, the XCM Sitemanager still has one major drawback. CMServer provides an XML format for each object in its database. To generate the layout for this data, a user of XCM Sitemanager is forced to program an XSL stylesheet for each object in the database. This XSL template is then applied to the XML representation of the object and the desired output format (e.g. HTML) is created. Even small websites usually contain more than a dozen different objects! Furthermore, XSL is a functional programming language with a quite complicated syntax. It would simplify the process of generating a website further, if the XSL programming was avoided and the layout of the data could be defined with graphical a user interface. The goal of the following thesis is to provide such a GUI.
# Contents

1 Objective of this Diploma Thesis 13

2 Introduction 15

2.1 Content Management with the CMServer 16

2.1.1 A Short Introduction to Content Management 16

2.1.2 The Architecture of CMServer 16

2.1.3 Creating a Website With CMServer 17

2.2 The XCM Sitemanager 18

2.2.1 The Concept Behind XCM Sitemanager 18

2.2.2 Creating XSL Templates With XCM Sitemanager 19

2.3 Objective 20

3 An Overview on Existing Graphical XSL-Editors 23

3.1 Excelon Stylus Studio 23

3.2 XMLSpy Stylesheet Designer 25

3.3 IBM Websphere 27

3.4 Conclusion 28

4 Requirements for a Graphical Template Editor 31

4.1 Requirements 31

4.1.1 The Complexity of XSL Must Be Hidden 31

4.1.2 Use the Metadata of the Data Model 31

4.1.3 Support for Every Base Data Type 32

4.1.4 Handling of Unknown Data Types 32

4.1.5 Extensible Architecture 32

4.1.6 Editing and Positioning of CMComponents 32

5
CONTENTS

4.1.7 Smooth Integration into XCM Sitemanager .................................. 33
4.2 The Architectural Concept .......................................................... 33

5 Implementation of the Frontend ...................................................... 35
5.1 The Look and Feel ................................................................. 35
5.2 Implementation of the Main Panel .............................................. 36
5.3 Implementation of the Structure Tree ......................................... 38
5.4 Representation of the Webpage .................................................. 38
5.5 Handling User Input .............................................................. 43
5.5.1 Implementing Drag&Drop ..................................................... 43
5.5.2 Editing the Webpage .......................................................... 44

6 Implementation of the Backend ..................................................... 49
6.1 Creating an XML Representation of a Page .................................. 49
6.2 The DTD of the XML Layout Representation .............................. 50
6.3 The XMLLayoutGenerator ....................................................... 55
6.4 Generation of XSL Templates .................................................... 57
6.5 Loading Existing Templates ...................................................... 60
6.6 Extending the Editor .............................................................. 60

7 Conclusion ................................................................................. 63
7.1 Fulfillment of Requirements ...................................................... 63
7.2 A Critical Review of the Implementation ..................................... 64
7.3 Future Enhancements and Possible Extensions ............................. 64

A Setup of XCM Sitemanager .......................................................... 67
A.1 Installation ............................................................................. 67
A.2 Starting and Configuring XCM Sitemanager ............................... 67
A.3 Required Software .................................................................... 68

B A Short Tutorial on the Graphical Template Editor ......................... 71
B.1 Configuring XCM Sitemanager .................................................. 71
B.2 The Layout GUI Tab ................................................................... 72
B.3 Loading an Existing Layout ...................................................... 73
B.4 Editing a Component .................................................................. 73
C.3.9 layout.webpage.WebpageDefaultComponent .................. 102
C.3.10 layout.webpage.WebpageDefaultContainer ................. 104
C.3.11 layout.webpage.WebpageMixedContainer .................. 105
C.3.13 layout.webpage.WebpageModel ............................. 108
C.3.15 layout.webpage.WebpageObject ............................ 110
C.3.16 layout.webpage.WebpageObjectFactory .................... 113
C.3.17 layout.webpage.WebpageObjectPopupActionHandler ...... 113
C.3.18 layout.webpage.WebpagePictureComponent ............... 114
C.3.19 layout.webpage.WebpagePictureLinkComponent .......... 115
C.3.20 layout.webpage.WebpageSitemapComponent ............... 117
C.3.21 layout.webpage.WebpageTextComponent .................. 118
C.3.22 layout.webpage.WebpageTextLinkComponent ............. 119
C.4 layout.dnd ....................................................... 121
C.4.1 layout.dnd.TransferableWebpageContainer ................. 121
C.4.2 layout.dnd.TreeDragGestureListener ........................ 122
C.4.3 layout.dnd.TreeDragSourceListener ....................... 122
C.4.4 layout.dnd.WebpageDropTargetListener ................... 123
C.5 layout.stylesheet ................................................ 124
C.5.1 layout.stylesheet.XMLLayoutGenerator .................... 124
C.5.2 layout.stylesheet.XMLLayoutLoader ....................... 125
C.6 layout.properties ............................................... 126
C.6.1 layout.properties.CMContainerPropertyToolbar .......... 126
C.6.2 layout.properties.CMDefaultPropertyToolbar .......... 126
C.6.3 layout.properties.CMMixedPropertyToolbar .......... 127
C.6.4 layout.properties.CMNavigationPropertyToolbar .......... 128
C.6.5 layout.properties.CMPictureLinkPropertyToolbar .... 129
C.6.6 layout.properties.CMPicturePropertyToolbar .......... 130
C.6.7 layout.properties.CMSitemapPropertyToolbar .......... 131
C.6.8 layout.properties.CMTextLinkPropertyToolbar .......... 132
C.6.9 layout.properties.CMTextPropertyToolbar .......... 133
C.6.10 layout.properties.ColorchooserDialog ............... 134
List of Figures

2.1 Rendering a webpage with the CMServer ........................................ 17
2.2 The XCM Sitemanager ................................................................. 19
2.3 Creating XSL templates with the XCM Sitemanager .................... 20
3.1 The eXcelon Stylus Studio .......................................................... 24
3.2 Altova’s Stylesheet Designer ....................................................... 25
3.3 The Authentic View of XMLSpy ................................................ 26
3.4 The Stylesheet Editor from IBM .................................................. 27
5.1 The Graphical Template Editor as a part of XCM Sitemanager ...... 35
5.2 The Main Panel of the Graphical Template Editor ....................... 37
5.3 The structure of the package webpage ........................................ 41
5.4 Resizing a WebpageContainer ................................................... 46
5.5 The CMTextPropertyToolbar ..................................................... 47
6.1 Table layout as visualized by Front Page .................................... 50
6.2 Creating the Table Layout ............................................................ 56
A.1 The Preferences Dialog .............................................................. 68
B.1 The Languages Dialog ............................................................... 71
B.2 The Layout GUI Tab ................................................................. 72
B.3 The Buttons of the Graphical Template Editor ......................... 73
B.4 Loading a Layout .............................................................. 73
B.5 Loading a Layout .............................................................. 74
B.6 Creating a new Layout .............................................................. 76
Chapter 1

Objective of this Diploma Thesis

Graphical Template Editor for Web Content Management

The challenge of web content management continues to rise as the demands of developers increase in terms of flexibility, generality and performance. These demands include support for universal client access, user customization, context dependent rendering, multilingual sites and on-line system evolution. Within the OMS project, we have developed an extensible web content management system (XWCM) that satisfies these requirements of web content management.

Since the development of a website involves working directly with the XWCM database at the level of individual applications, component and template objects and the associations between them, a graphical authoring tool (CMSGui) has been developed. In its current version CMSGui assists the user in defining the schema of a website, creating data objects, structuring the content and publishing pages. It does however not alleviate the job of defining the XSL templates necessary to render a page for multiple presentation channels.

The goal of this thesis is to develop a graphical template editor that enables a user to define the layout of a page (container) graphically. Based on this graphical definition the editor then creates the necessary templates and stores them in the database. To maintain correct functionality of a website, the templates have to be associated with the appropriate content objects. In addition of this basic functionality, it should also be possible to define page templates for list and table based layouts. The editor should be integrated within the current version of CMSGui in an easy to use way. The main tasks of this diploma thesis are as follows:

- Investigation of the current XWCM framework and the CMSGui graphical user interface.

- Survey and feature analysis of existing graphical XSL editors to compile a list of desirable and state-of-the-art functionality.

- Analysis of XSLT and its capabilities. From this analysis, a specification of features and the functionality of the new editor should be created. This should also include designs of the user-interface and a summary of the used components and libraries.
CHAPTER 1. OBJECTIVE OF THIS DIPLOMA THESIS

- Specification of the required extensions to the XWCM schema to store the new metadata necessary to the graphical template editor.

- Implementation and integration of the new XSLT editing tool and testing by means of a real-world website.

The project report should clearly describe the design of the user-interface and the architecture of the editing tool. Further it should show how it was integrated to work with the existing parts of WXCM and CMSGui. A detailed description of any modifications to existing software, schema changes in XWCM and newly defined formats should also be included in the report. Last but not least, there should be an installation and user-guide to help new users setting up the system and working with it.
Chapter 2

Introduction

The number of large websites and webapplications on the internet has grown rapidly in the last few years. Today enterprises, universities, governments, schools and other large organisations such as sports clubs or foundations provide very large websites to their customers or members. The management of the vast amount of information, which is contained within these sites is a great challenge for every kind of organisation. Especially the following issues arise:

- Information should always be up to date.
- Certain parts of a website should be reusable, e.g. the logo of the organisation.
- It is difficult to standardise the publishing process for information within an organisation.
- It should be possible to search for specific information within a large website. HTML is not suitable for this.
- HTML code leads to a bad separation between content and presentation.
- Normal HTML Code is static.

For a detailed discussion of these problems please refer to [1]. However, all these inadequacies of normal HTML-based websites have lead to the development of various so called content management systems. A detailed overview on content management can also be found in [2]. In this thesis we will focus on a specific solution, the CMServer content management system developed at the Institute for Information Systems at the Swiss Federal Institute of Technology (ETH Zürich) [2][3]. The next section gives a short overview on the functionality of the CMServer. We will also show, that it is quite complicated and error-prone to develop a website for the CMServer without any tool, which assists this task. For this reason, the XCM Sitemanager, a graphical authoring tool for the CMServer was developed [3]. The XCM Sitemanager will be discussed later. In particular, we will resent how it lacks a sufficient support for the creation of layout of the given data. This leads directly to the objective of this thesis, which ends this chapter.
CHAPTER 2. INTRODUCTION

2.1 Content Management with the CMServer

2.1.1 A Short Introduction to Content Management

Content management systems focus on the following tasks [2]:

- Reusability of parts of the website (mainly images).
- Support for multiple output channels (e.g. HTML and WML).
- Support for content in different languages.
- Standardised publishing processes for new data.
- Improve the scalability of websites.

Classically, there are two different approaches to content management. The first is typically taken by people who worked as webmasters or webdesigners. They focus on the website as a collection of images, texts, tables and links. Therefore, the database used in such a content management system will contain types like table, text and image. It will support the generation of websites out of these objects through scripting or by the use of templates. [2] calls this kind of content management the document-based approach. The other approach, called the data-based approach [2], is generally chosen by people who have experience with databases and the management of large information-systems. In this approach, the heart of the content management system is a database containing arbitrary data-types. To publish objects from the database, the database is queried for the needed objects and the results are then used to generate the desired output (e.g. a webpage). For details on the two approaches refer to [2]. The CMServer content management system is a synthesis of these two paradigms. In the next section, we will see how it works and how it makes use of the advantages of both system without inheriting their drawbacks.

2.1.2 The Architecture of CMServer

As mentioned above, the CMServer content management system is a "middle course" [2] between the document-based and the data-based approach. It defines a few basic web data types as they are often found in document based content management systems. But at the same time, the user is allowed to generate additional data types and the system provides great flexibility for any kind of extension that may be needed. The CMServer is a server-side application based on the Java servlet architecture. It stores its data in a database, which implements the object-oriented data management framework OMS (Object Model System)[4], developed at the Institute for Information Systems at the ETH Zurich. At the moment OMS Java [5] is used.

As we already stated, the database contains a few base types. The most basic type is CM-Component, it represents an arbitrary part of any document prepared for publishing, normally this document is a webpage!

A subtype of CMComponent is CMContainer. CMContainer is the supertype for all data types which can contain other CMComponents. Other data types are for example CMText for textual data, or CMPicture for images. Each object in the database may have different contents for different languages.
The database also contains types to facilitate the standardisation of the publishing process. These types are used by the CMServer component to control who has permission to change certain content of a website. A user of CMServer can also incorporate his own subtypes of CMComponent into the model. This allows a great amount of flexibility. We will not go into details here, as the data model behind the CMServer is detailed [2]. In order to work properly, the CMServer needs a webserver with a container for Java servlet execution. An example of such a webserver is the tomcat webserver of the jakarta project. For details on the tomcat webserver visit [6].

The best thing to show how the CMServer works is an example: Assume a browser sends a http request for a file called index.html to a website managed by the CMServer content management system. The CMServer servlet is called to handle the request. The servlet determines which objects in the database are needed to render the desired webpage in the correct language. Then, the servlet recursively gets an XML representation of these objects from the database. Along with the XML file, the servlet also gets an XSL template for each object, containing layout instructions for the desired output channel (in our case HTML). The Xalan XSL processor is invoked to generate the requested index.html from the XML representation of each object and the corresponding XSL template. Finally, the resulting file is sent to the client browser. This process is shown in figure 2.1.

2.1.3 Creating a Website With CMServer

In the example above, we have shown how CMServer works. What remains to be explained is how a website managed by the CMServer is created. As we will see, this is a
quite difficult and tedious job. First, a correct database setup is needed. A new database
is created in the OMS JAVA system. Then all the data types, their constraints and associations
among each other must be declared. Of course, for the base CMServer data model, a
corresponding data type definition (ddl) [4] exists. If user-defined data types are required,
these must be declared as well.
Now the content of the database has to be generated. This means, instances of certain
type, such as text, pictures or links, are created. If multiple languages are supported, every
language needs to be represented by a different content object. Then, the structure of the
website is declared by defining the correct associations. In other words, it is defined which
container holds which text, picture, link etc.. This part of the task also includes the creation
of the correct directory structure of the website. Last but not least, for each supported output
channel, XSL templates containing the layout instructions must be programmed and
associated with the corresponding data type in the database. For detailed review of the
complexity of the described tasks refer to [3]. In order to bring some structure into the
described process and to simplify it, the XCM Sitemanager was developed. The following
sections describes this application shortly.

2.2 The XCM Sitemanager

XCM Sitemanager is an application that greatly eases the creation of a website for the CM-
Server content management system. It was developed as a Diploma Thesis at the Institute
for Information System at the ETH Zurich [3]. A screenshot of the User Interface of the
XCM Sitemanager is shown in figure 2.2.

2.2.1 The Concept Behind XCM Sitemanager

The graphical user interface contains a panel with six tabs. These tabs include Model,
Object, Structure, Layout, Layout GUI and Test. Together they represent the basic idea
behind the XCM Sitemanager, which is to divide the creation of a website into five steps.

1. Model-Generation:
   As described above, the first step is always to create the data types needed by the
   website. The first tab provides that function. This includes the tedious task of creating
   all the needed associations and collections.

2. Object-Creation:
   The Object tab allows the user to create instances of each data type and to add language-
   dependent content to these instances.

3. Site Structure:
   On the Structure tab, the user defines the structural hierarchy of the website. Here its
directory structure is defined and pages to are added the directories. Then the pages
   and other containers are filled with their objects.

4. Presentation:
   A website has a layout. As we have seen in the description of the architecture, the
   CMServer defines the layout of webpages by XSL templates assigned to each of the
   objects in the database. In the fourth tab, the user creates such XSL templates and
2.2. THE XCM SITEMANAGER

assigns them to the corresponding object. The fifth tab seen in figure 2.2 labelled "Layout GUI" was not present in the original version of XCM Sitemanager. It provides a graphical XSL template editor. We will describe this tab in detail, as it is the main focus of this diploma thesis.

5. Compatibility:
CMServer is still under construction. Therefore, in order to be reusable with newer versions of the CMServer, it is very important that websites remain independent of the current version. The last tab provides a few tests to ensure that.

![Figure 2.2: The XCM Sitemanager](image)

Besides the described functionality, XCM Sitemanager also allows to import already created databases, it provides the possibility to set the languages supported by the website and makes other small but important tasks more easy. Everyone who has created a website with the XCM Sitemanager will agree, that it really facilitates the working with CMServer. But there is still a shortcoming, which we will describe in the next section.

2.2.2 Creating XSL Templates With XCM Sitemanager

As a proof of concept the XCM Sitemanager was a success. Although its target was to simplify the creation of a website for the CMServer content management system, Yet, some jobs can still be made a lot more easy. One such difficult and time-consuming task is the programming of the XSL templates needed for each object in the database.

Let’s have a look at the support XCM Sitemanager provides for this task. Figure 2.3 shows the tab to create and assign XSL templates to the different objects. The tree in the left window of the application displays the structure of the website. Each node in the tree represents an object in the database. The top panel labelled with “Object” shows the XML representation of the selected object. There are two kind of templates. The default template is normally used to render the a CMComponent. But sometimes, the CMComponent must be rendered differently depending on its context. In this case the context template is
CHAPTER 2. INTRODUCTION

Figure 2.3: Creating XSL templates with the XCM Sitemanager

applied. The context of an object is the container it is assigned to. The comboboxes on the right of the textpane allow the user to assign the template to the object, either as context or as default template. The templates are edited in the textpane labelled "Templates". The template’s name can be entered in the bottommost textfield. As soon as the template is saved, it is possible to assign it to an object as described above. The textpane used to enter the template’s code has no features like syntax-highlighting or pretty-printing of the code. As we see the support for creating of templates is very limited.

The website in our example is small. It only contains a few pages with not more than 9 objects per page. Nevertheless, to render every webpage correctly, the developer of the website needs to program more than 20 different XSL templates. Then he must assign each of them to the correct object. The template-editor offers no support for programming, so in practice templates are created with another editor, then the code is inserted with copy and paste. Clearly, this is not a suitable solution. Without a powerful environment to create the layout of the website, XCM Sitemanager loses a lot of its practical use! The shortcomings described in this section lead directly to the task of this diploma thesis, which we will describe next.

2.3 Objective

As seen above, the XCM Sitemanager lacks a suitable method to create the layout of webpages. Modern webpage editors provide graphical user interfaces to develop webpages. In order to be useful, XCM Sitemanager needs such a tool as well. Therefore, the objective
of this thesis is to develop a graphical template editor to that allows define the layout of a webpage visually. Based on this graphical definition the editor then creates the necessary templates and stores them in the database. To maintain correct functionality of a website, the templates have to be associated with the appropriate content objects. In addition of this basic functionality, it should also be possible to define page templates for list and table based layouts. Last but not least, the editor must be integrated smoothly into the current version of the XCM Sitemanager.
Chapter 3

An Overview on Existing Graphical XSL-Editors

The W3C XSL recommendation is quite new. Nevertheless there are already many graphical editors for XSL on the market. In this chapter we will shortly review three of these tools. This market-analysis should help to get a better overview on the possibilities of XSL, and on the functions that a graphical template editor must provide. We will compare the tools in terms of their functionality and their usability. As the objective of this thesis is to create a graphical template editor for the CMServer, we will also discuss whether these tools could be used.

3.1 Excelon Stylus Studio

The Excelon Stylus Studio was developed by the eXcelon Corporation [7]. According to their website, the programm offers a “XML to HTML WYSIWYG (What see is what you get) Editor”. A demo version of the software which is limited to thirty day is available on the website.

The Frontend Figure 3.1 shows the user interface of the eXcelon Stylus Studio. It is divided into six parts, which are labelled with green arrows in figure 3.1. The large section in the center is filled by a representation of the webpage to be created. In this representation the user can generate the layout of the XML file. The top left part of the GUI contains a tree with the project structure. Below the project structure is a window with the properties of the currently selected object. The upper part of the interface is filled by a menubar with tools to edit the layout of a webpage. To the right side of the webpage representation is a tree view of the structure of the XML document. Last but not least, the bottom of the frontend is filled by a window containing a preview of the created webpage.

We summarize the functionality of the programm in the following. If data from the XML file is to be inserted, one selects the desired element or attribute in the tree. Via Drag&Drop this selection is moved into the webpage. When the drop occurs, a dialog pops up and asks for the type of XSL instructions that should be applied to the selected data. Depending on
the type of element or attribute, different XSL instructions are available. The properties menu allows to generate path expressions to specialize the selection even more. The GUI provides support for HTML elements like lists, tables or links, also its possible to insert pictures and to modify the fonts and colors for each element.

Creating a webpage The first impression of the application is not bad. It is easy to understand and simple webpages are generated quickly from small XML documents. But it is not possible to create a more complicated webpages, not to mention about whole websites, without programming XSL code. The Stylus Studio can assist the creation of large websites with its graphical editor, but it cannot be used to do it completely. The reason for this is found in the concept behind the Stylus Studio. The tree view of the XML document shows only the structure of the XML file, not the actual data contained within it. Therefore, it is very difficult to identify the correct element and to place it on the right position within the page. In order to do so, one must first place the element on the screen and then adjust the path expression to select the desired element or attribute value. This problem not only arises with the XML data provided by the CMServer, it occurs with any large XML file. Therefore, it is often faster to program the required XSL code. Nevertheless, one must admit that the combination of the normal XSL editing functions of the Stylus Studio with the WYSIWYG editor is a reasonable solution to generate even large websites.
3.2 XMLSpy Stylesheet Designer

One of the most popular XML editors is XMLSpy from Altova [8]. XMLSpy itself does not contain a graphical XSL editor. But Altova offers an additional tool, the Stylesheet Designer which can be integrated with XMLSpy. A demo-version limited to thirty days is available on Altova’s website.

![Figure 3.2: Altova’s Stylesheet Designer](image)

**The Frontend** Figure 3.2 shows the user interface of the Stylesheet editor. Again, it is divided into different parts. The top of the interface is filled with a menubar providing different tools to edit the layout of a webpage. There are buttons to insert pictures and links or to create and edit lists and tables. On the left side of the window, just below the menubar, is a tree displaying the structure of the currently edited XML file. The bottom-most windows on the left side are used to edit the properties of the currently selected objects on the page view. The largest part of the window is filled by the representation of the webpage being created. A special feature of the Stylesheet Designer is, that it outlines the structure of the page by showing markers to indicate the positions of the elements and attributes within the page. Markers are arrows labelled with the names of the elements or attributes they represent. If the label of the arrow contains a “@” character, the arrow represents an attribute, otherwise it is an element. An arrow directing to the right marks the beginning of an element or attribute position, an arrow to the left indicates the end. On the bottom of the page view the user can switch to a preview window or inspect the stylesheet’s XSL code.
The Stylesheet Designer has a concept that differs from the Stylus Studio. The only similarity is, that elements or attributes can be placed with Drag&Drop on the desired location within the page. But unlike the Stylus Studio, there is no possibility to generate XPath expressions to further specify the selected elements or attributes. All elements or attributes of the same type are just enumerated on the webpage. The tables and lists created with the Stylesheet Designer are adjusted automatically to support this behaviour. For example, to create a list of all elements of the same type, it suffices to drag one of the desired elements into the list. In the generated HTML code the list has as many items as there are elements!

Creating a webpage  The Stylesheet Designer is not suitable to create a real webpage, as there is no possibility to create XPath expressions. The Designer not even contains a simple XSL editor to produce the needed queries. To do so, one must first create a stylesheet with the designer and then adapt it with the XSL editor of XMLSpy. Clearly, this is not a comfortable way of creating a website. As we see it, the Stylesheet Editor is not intended to develop websites. It is better used to create data sheets or input masks used within websites. This impression is also supported by the way Altova positions its Stylesheet Designer on the market. According to the product’s website [8], the Stylesheet Designer allows people with no or only little experience with XML and XSL the creation of good looking HTML pages. This strategy is also observable within the XMLSpy application. To add data to existing XML files, XMLSpy offers a special so-called authentic view. This view is based on the stylesheet created in the Designer. The whole complexity of XML and XSL is hidden from the user. Figure 3.3 shows this authentic view.

Figure 3.3: The Authentic View of XMLSpy
3.3 IBM Websphere

In autumn 2002 IBM released a technology preview of a graphical XSL editor pluggable to its huge software development solution Websphere. A trial version of Websphere is available on IBM’s webpage [9]. The stylesheet editor is downloadable for free at [10]. A detailed description of the stylesheet editor is found in [11].

The frontend The user interface, shown in figure 3.4 is structured as follows. The top of the window is filled by a menupanel with tools to edit the webpage. Besides the usual buttons to create tables and lists and to edit fonts and colours, there are also tools to easily create XSL expressions like XPath queries. The classical tree view of the XML file is found in the top left part of the interface. Unlike the other two products, IBM does not just show the structure of the XML file but the concrete elements and attributes with their values. Below the tree lies a window which contains all rules (templates) used in the currently created stylesheet. The center of the GUI is the page view, where all the editing takes place. Instead of showing an empty page when a new XSL stylesheet is created, IBM reverses the commonly used principles and shows every element contained within the XML file on the initial page. Elements are chosen with mouse clicks or by selection in the structure tree. A combobox in the menubar above shows XSL expressions, which fit to the selected element. For example, it is possible to select only this type of element, or even more specific this element itself. With the template rule editor window in the bottom left of the user interface, the user can customize the XSL expression as required. When a special element or group of elements is selected, the editor allows to define the formatting of this element.
CHAPTER 3. AN OVERVIEW ON EXISTING GRAPHICAL XSL-EDITORS

For example, you can edit the font style or add the element to a list.

Creating a Website  The stylesheet editor is clearly the best of the three presented tools to create websites. Especially the fact, that the whole data within the XML file is accessible and specialized queries can be created for every single element, allows the creation of large websites also for the CMServer. A good feature is the graphical XSL expression designer, it eases the creation of the often tedious to program XPath expressions significantly. Also very important, the integration with the common textual XSL editor works absolutely fine. It is no problem to fine tune the XSL code with the textual editor after the framework of the webpage was created with the stylesheet editor. All these points together make this program a great tool to create webpages or even whole websites. The only disadvantage is, that it requires the user to have a good knowledge about XML and XSL.

3.4 Conclusion

In this chapter we discussed three different approaches for the same problem, how can one simplify the creation of XSL stylesheets. All three tools share the problem that they must support every possible XML file. Of course, this leads to great difficulties because its nearly impossible to create a graphical editor without at least some knowledge about the semantics of the supported XML files. As we saw, the resulting programs differ very much in functionality and handling. The main problem of Stylus Studio and XMLSpy is, that they do not show the actual data contained in the XML file to the user. The XMLSpy goes even one step further, it totally blocks the possibility to use XPath expressions to make specialized selections of certain elements. On the other hand, XMLSpy is the only tool that does not expect the user to be familiar with XSL. IBM’s stylesheet editor represents the other extreme, it gives the programmer total control over the creation of the stylesheets, but this requires knowledge about XML and XSL.

As we have seen, the creation of XSL files for our example is sometimes quite tedious with the tools introduced above. Note that none of the presented tools is completely suitable to solve the problem of this diploma thesis, as they are developed to create XSL stylesheets for arbitrary XML formats. We have the great advantage, that the format and the structure of the XML data to be transformed is already known. Of course, this makes the problem more specific and therefore more easy.

However, as a conclusion of this short market analysis, we propose the following points as important for the development of our graphical template editor:

- The data contained within the XML file, for which we create a stylesheet should not be hidden from the user. The schema or DTD (Document Type Definition) of the XML file is not sufficient when it comes to larger webpages or whole websites.

- A good editor should hide as much as possible of the XSL code from the programmer. But on the same time, if the programmer really wants to modify the automatically created stylesheet, this must be possible as well.

- It is a great advantage if the graphical stylesheet editor has some knowledge on the semantics of the XML files it must create stylesheets for.
3.4. CONCLUSION

In the next chapter, we will use these criteria to define the requirements of our graphical template editor.
CHAPTER 3. AN OVERVIEW ON EXISTING GRAPHICAL XSL-EDITORS
4.1 Requirements

4.1.1 The Complexity of XSL Must Be Hidden

XSL is a very powerful functional programming language. It is however, also quite com-
plicated. XSL documents must be wellformed and valid XML documents. We will not
discuss whether this is a good approach or not. Fact is, a lot of people, especially webde-
signers and webmasters who are the target user group of XCM Sitemanager, will regard
the syntax of XSL as too complex. Therefore, a graphical template editor must totally hide
the XSL programming language from the user. This point is crucial, as the previous chapter
already showed that graphical template editors become useless, when XSL is not fully hid-
den. Of course it still must be possible for expert XSL programmers to edit their stylesheets
manually.

4.1.2 Use the Metadata of the Data Model

It is absolutely necessary to use the metadata contained in the data model of the CMServer.
This is one great advantage of the CMServer content management system compared with
other solutions. Consider the association between a CMComponent and a CMContainer:
The knowledge, that a certain CMComponent lies within a given context greatly simplifies
the creation of that CMComponent. A webpage itself is nothing else than a CMContainer holding other CMContainers and CMComponents.

### 4.1.3 Support for Every Base Data Type

The graphical editor must produce reasonable output for every base data type defined in the CMServer schema, that can be placed on a webpage. The base types to be supported at the moment are:

- **CMText**: This type represents a simple text without markup.
- **CMLink**: A CMLink represents a link on a webpage.
- **CMUrl**: This type simply contains an URL.
- **CMPicture**: Represents an image.
- **CMCollection**: A normal CMContainer that contains other CMComponents.
- **CMMixed**: A CMMixed can contain text with markup and other CMComponents in addition.
- **CMSitemap**: Contains a map of the structure of the website.
- **CMNavigation**: Represents a navigation in a webpage.

### 4.1.4 Handling of Unknown Data Types

One of the great advantages of the CMServer content management system is the extensibility of its data model. Of course, XCM Sitemanager must also support this extensibility. Therefore, in the presence of an unknown (user defined) data type, the graphical template editor should not crash, instead it should apply a default template. This template can be removed to customize the layout of the user defined type.

### 4.1.5 Extensible Architecture

This requirement is closely related to the handling of unknown data types mentioned above. Because the CMServer data model is extensible, it is possible that in future versions new data types extend the set of base types. In this case, it must be possible to extend the graphical editor in order to provide support for these new data types.

### 4.1.6 Editing and Positioning of CMComponents

In order to provide a flexible layout of the webpages, a CMComponent must be placeable everywhere within its context. Additionally, it must be possible to set properties like fonts, colours and backgrounds for every CMComponent.
4.2. THE ARCHITECTURAL CONCEPT

4.1.7 Smooth Integration into XCM Sitemanager

One of the reasons XCM Sitemanager was originally developed, is to facilitate the creation of websites for the CMServer. In order to go one step further on this path, the graphical template editor should provide a intuitive and easy-to-use interface. To avoid any disruption at the impression a user gets using XCM Sitemanager, each part of it must have a similar handling. This means that the graphical template editor must inherit as much as possible of the look and feel from the already existing parts of the program.

4.2 The Architectural Concept

Out of the above explained requirements, we now derive the architectural concept of the graphical template editor.

A close look on how the graphical template editor works, reveals that the program is actually a graphical XSL compiler. Clearly, the output of the tool is a set of XSL templates which have to be computed from a graphical user-defined representation of the webpage.

Generally, a user wants to create these templates as follows:

1. Out of a collection of all pages within the site, the user chooses a webpage whose layout will be defined.
2. Then the layout is created by dragging and dropping CMComponents on the screen and by setting properties like font, colour and background.
3. When the layout is satisfactory, the user hits save-button and the editor starts creating the templates for the different CMComponents.
4. As soon as the editor has finished the generation of the templates, it assigns them to the correct CMComponent.
5. After finishing the creation process the editor shows a message to the user.
6. The user now may starts to edit the next webpage.

As we see, there are clearly two different tasks. First allowing the user to draw a layout and second creating the XSL templates. It is therefore straightforward to divide the graphical editor into a frontend and a backend. The frontend is responsible for the drawing of CMComponent objects on the screen and for the reaction on user input. The frontend should consist mainly of subclasses of existing Java-Swing components. This facilitates the creation of the user interface because the powerful Swing architecture can be exploited to react on user input and to draw on the screen.

The other component, the backend, is hidden from the user by the frontend. The backend, invoked when the save-button is pressed, creates the XSL code. In order to do this, an algorithm to convert the layout drawn by the user into a machine-readable description must be implemented. The description should be stored in a data-structure suitable to create XSL templates. It turned out to be a good idea, to use an XML representation as description of the desired layout of every CMComponent. In a second step, an XSL stylesheet is applied to this XML description to generate the desired XSL templates. This two-step approach is easier and more general than creating XSL templates directly. In the next two
chapters, we first explain the architecture of the frontend and then show how the backend is implemented.
Chapter 5

Implementation of the Frontend

5.1 The Look and Feel

This chapter gives an overview of the appearance of the graphical template editor and how a webpage is edited. The main window of the graphical template editor is a JPanel. Like any other part of XCM Sitemanager (e.g. model or structure), this JPanel was added to the tabbed pane of XCM Sitemanager. The tab is labelled Layout GUI. The appearance of the graphical template editor is shown in figure 5.1. Most of the JPanel is filled by the page being edited. In figure 5.1 one can easily recognize the page as the large area with texts and images filling most of the bottom right part of the picture. The left part of the JPanel is filled with a tree. While editing a webpage, this tree shows the structure of the webpage.

Figure 5.1: The Graphical Template Editor as a part of XCM Sitemanager
under construction. If no page is edited, the tree shows the structure of the whole website and allows the selection of webpages to be edited. The top of the JPanel is divided into two different parts. The left side contains general buttons to enable loading and saving of layouts, or to create or cancel a new layout for a webpage. The right part shows a floatable toolbar containing buttons and comboboxes to edit the properties of the currently selected CMComponent. The selected CMComponent in figure 5.1 is a CMText, therefore the property toolbar shows comboboxes to change the font size, font family, font color and the background colour.

Initially the webpage is empty. The placing of CMComponents on the webpage is achieved by Drag&Drop. CMComponents whose parent is the webpage itself, can be selected in the tree and then be dragged to the desired place on the webpage. If a CMContainer is placed, all its children are drawn automatically on default positions within the container. The currently selected CMComponent is covered by a semi-transparent red shade. The selection works by clicking on the CMComponent, or by selecting it in the tree. Each Representation of a CMComponent has a border. The CMComponent is resized as usual by dragging this border with the mouse. CMComponents are moved around their container by dragging them with the mouse.

5.2 Implementation of the Main Panel

The main panel is implemented by a class called XSLGUIPanel. Technically, it is not very complicated. As mentioned before the XSLGUIPanel is divided into three parts. The top of the panel contains the tools to edit the webpage. The left part contains a tree holding the structure of the webpage. The rest of the panel is filled with a JScrollpane that contains the webpage under construction. The layout manager used is GridBagLayout because it is very flexible. Figure 5.2 gives an overview of the main panel pointing out its different subcomponents. In the following the most important methods of XSLGUIPanel are explained.

private void initWebAppStructureTree()

This method loads the structure tree of the whole website. It is called at the startup of the application and when the user finishes editing a webpage by clicking the cancel button.

public DesignGUITree initPageStructureTree(WebpageModel webpageModel)

This method loads the structure tree of the webpage being edited. The structure tree is represented by the class DesignGUITree which will be explained in the next section. The method is called when the user selects a webpage to create a layout. This method also assigns the mouse listeners and the tree selection listener to the DesignGUITree.

private void initWebpage(String templateName, boolean fixedWidth, int webpageWidth)

This method is called when the user wants to edit a webpage. It loads the JPanel representing the webpage and makes sure that every thing is initialized correctly.
public void setPropertyToolbar(JPanel propertyToolbar)

Sets the actual property toolbar. The property toolbar is used to edit the properties of the CMComponents within the webpage. Every type of CMComponent has a different property toolbar. See the section 5.5 for details.

jButtonLoadActionPerformed(java.awt.event.ActionEvent evt)

Invoked when the load button is pressed. Loads the layout currently assigned to the selected webpage. This method only calls the class XMLLayoutLoader to do the job.

private void
    jButtonCancelPageEditActionPerformed(java.awt.event.ActionEvent evt)

Invoked when the cancel button is pressed. Removes the actually edited webpage, makes sure that everything is finalized correctly and loads the structure tree of the website.

private void

This method is called when the user clicks the new layout button. It shows a dialog asking for the name of the webpage layout and if the webpage has a fixed width. When the dialog is finished, the method calls initWebpage to do the rest.
private void
    jButtonSaveActionPerformed(java.awt.event.ActionEvent evt)

Called when the actual layout should be saved. It displays a warning message that the old
layout will be overwritten, then it erases all old associations and invokes the creation of the
layout with the command: this.page.createXMLLayout().

For more information read the following sections and take a look at the API documenta-
tion in appendix C.

5.3 Implementation of the Structure Tree

The structure tree is represented by the class DesignGUITree which extend JTree. It shows
the structure of the webpage currently edited. The node objects of the structure tree are
GUIObjects or GUIContainers. These two types were developed originally to represent
CMComponents in XCM Sitemanager [3]. They offer a method to access the correspond-
ing CMComponents in the database. Therefore they were used in this tree as well. The
DesignGUITree contains an inner class DesignTreeCellRenderer that implements the Tree-
CellRenderer interface. The DesignTreeCellRenderer contains the information how to draw
the different types of nodes. For example, a node object of type GUIContainer is repre-
sented by a box-icon. A node object of type GUIObject by a simple withe rhombus. Lets
have a look at the most important methods of DesignGUITree.

private void loadPage()

Loads the whole tree, starting with the root (the page). This means the children of the
GUIContainer representing the webpage are traversed and for every GUIObject a tree node
is created and added to the root node. If the GUIObject is a GUIContainer its children are
traversed as well.

loadNode(DefaultMutableTreeNode node)

This method starts with the given node and is called recursively to reload the tree. Its
invoked by loadPage to do the actual work.

public void selectGUIObject(GUIObject object)}

This method selects the node which represents the given GUIObject.

public Enumeration getChildren(GUIObject object)}

Returns the children nodes of the given GUIObject.

5.4 Representation of the Webpage

The webpage and its CMContainers and CMComponents are represented by the classes of
the package ch.ethz.globis.cmserver.gui.layout.webpage (see appendix C). In the following, we
5.4. REPRESENTATION OF THE WEBPAGE

will show how this package is structured.
The class diagram of the package is displayed in figure 5.3. It shows the dependencies between the different classes of the package webpage, and its connection to the structure tree (the class DesignGUITree). Please note that the methods and variables were stripped from the model to improve the clarity of the diagram. As mentioned in the section above, the goal of the frontend implementation was to rely on existing Swing components as much as possible. In order to find out how a webpage is represented best on the screen, it is a good idea to inspect the data model and to find out if some Swing Components can be used to represent the data types in the model!

Basically, two different types of CMComponents appear on a webpage, containers and components. While containers hold other components or containers, components just contain text or images. The Swing class JLabel itself perfectly suits to components. Containers are represented best by the class JPanel which is a Swing container. Unfortunately, JLabel and JPanel do not cover every possible action with their methods. For example, every CMComponent has specific properties like font, colour and background, which should be editable. Furthermore, each CMComponent has a border and can be selected, dragged and resized. Especially dragging and resizing are difficult, because it is not desirable that a component may be outside its container or that it suddenly overlaps other components. These problems apply to every CMComponent, but the solution depends on the concrete type of the CMComponent. For example, while the moving of a CMContainer involves moving its children, moving a CMPicture only means moving a JLabel with an image on it. As we see, there is a set of necessary methods in each representation of CMComponent. Therefore the interface WebpageObject was defined. This interface holds the signature of methods every representation of a CMComponent on the screen (including the webpage) must implement.

**Methods of WebpageObject** Details on WebpageObject are found in the API (see appendix C). Nevertheless, we will shortly discuss the most important methods.

```java
public void drawContent(Dimension size, Point location);
```

Draws the WebpageObject on the webpage (the screen). This Method is responsible to set the position and size of the component, to get its content (e.g. the text or picture) and finally to display everything on the screen!

```java
public boolean relocateContent(Point newLocation, Dimension newSize)
```

Resizes the WebpageObject and moves it to the given location. This method fails when the WebpageObject cannot be placed on its new location because of the size or because of an intersection! Returns true if it is possible to resize the content.

```java
public Element createXMLLayout()
```

As we mentioned above, the layout of a WebpageObject is stored in a XML element. This method is called by the backend to create the XML element for the WebpageObject.
public void saveToDatabase(Element webpageObjectXMLTemplate, 
   String templateName)

This method associates the given XML layout representation with the CMComponent represented by the WebpageObject. It then calls `transformXSLStylesheet` to generate the XSL template. The template is saved to the database as well. This method is also called from the backend.

public String transformXSLStylesheet(String xml)

To transform the XML layout of the WebpageObject into the desired XSL template, this method is called by the method `saveToDatabase`. These are only a fraction of the methods defined in WebpageObject, but they are by far the most crucial in understanding the WebpageObject interface. For details on these and other methods refer to appendix C.

Next, we discuss the concrete implementations of WebpageObject. Again, we must distinguish between CMContainers and other CMComponents.

**CMContainers** For CMContainers, an abstract class `WebpageContainer` which extends `JPanel` and implements `WebpageObject` was introduced. In addition `WebpageContainer` contains special methods to manage the WebpageObjects which lie within this `WebpageContainer`. `WebpageContainer` is the superclass of all classes representing subtypes of CMContainer, at the moment these include:

- Webpage for the CMContainer with the function `page`.
- WebpageCollectionContainer for the CMContainer with the function `collection`.
- WebpageMixedContainer for the CMMixed type.
- WebpageDefaultContainer for unknown subtypes of CMContainer (at the moment).

**CMComponents** The other CMComponents which are not subtypes of CMContainer are represented by `WebpageComponent`, which extends `JLabel` and implements `WebpageObject`. At the moment these are:

- WebpageTextComponent for the type CMText.
- WebpagePictureComponent for the type CMPicture.
- WebpageNavigationComponent for the CMNavigation type.
- WebpageSitemapComponent for the CMNavigation type.
- WebpagePictureLinkComponent for the type CMLink with an image resource. It is needed to distinguish this class from the WebpageTextLinkComponent, which represents links with text resources. The reason is, that each content type of a link resource must be accessed with a different query.
- WebpageTextLinkComponent for the type CMLink with a text resource. The other type of link resource supported a the moment.
- WebpageDefaultComponent for unknown subtypes of CMComponent (at the moment).
5.4. REPRESENTATION OF THE WEBPAGE

Figure 5.3: The structure of the package webpage
CHAPTER 5. IMPLEMENTATION OF THE FRONTEND

The class WebpageModel  We have seen how the data types from the CMServer data model are represented on the screen. What remains to be discussed is how the different types of WebpageObjects are created, how the synchronization with the structure tree works and how the interaction with the user is processed. The solution to some of these problems is found in the class WebpageModel. Every Webpage has an associated WebpageModel which creates and manages its WebpageObjects and acts as its connection to the structure tree or to other parts of XCM Sitemanager. In the following we discuss some of its methods.

private void createWebpageObject(WebpageContainer parent,
DefaultMutableTreeNode parentNode);

This method is called at the begin of the editing session of a webpage. It traverses through the structure tree starting with the root node (the webpage) and creates for each CMComponent the corresponding WebpageObject. The method does not generate the WebpageObject itself. Instead it passes the current CMComponent to the class WebpageObjectFactory, this factory class returns the correct instance of WebpageObject according to the received CMComponent object. From this moment on, the WebpageObject is instantiated, but it not visible on the screen yet. In order to become visible, the WebpageObject must be placed on the screen. The placing of WebpageObjects is described later in this chapter.

public void selectionChanged(WebpageObject content)

Makes sure that the selection of a WebpageObject leads to the selection of the corresponding GUIObject in the tree.

public void setActualGUIObject(GUIObject actualObject,
GUIContainer parentObject)

This method is responsible that the selection of a GUIObject in the tree leads to the selection of the corresponding WebpageObject.

public GUIObject getActualGUIObject()

Returns the actually selected Node of structure tree. It is worth to point out that the node objects of the structure tree are of type GUIObject here.

public WebpageObject getActualWebpageObject()

Returns the actually selected WebpageObject.

public GUIObject getGUIObject(WebpageObject webObj)

Returns the GUIObject represented by the given WebpageObject.

public WebpageObject getWebpageObject(GUIObject obj)

Returns the WebpageObject represented by the given GUIObject.

This should suffice to understand how a webpage is represented in the graphical template editor. For a detailed insight into all methods of WebpageModel refer to the API documentation in appendix C. In the next section we describe the handling of user input.
5.5 Handling User Input

The description of the user input will be divided into two parts. First, the Drag&Drop functionality is explained. Drag&Drop is used to place WebpageObjects on the JPanel representing the webpage. Then, we describe how WebpageObjects are resized, moved around in their WebpageContainers and how their properties like font, colour and background are manipulated.

5.5.1 Implementing Drag&Drop

The graphical template editor uses Drag&Drop to place the WebpageObjects on the webpage. Initially the webpage is empty. If a user wants to place a WebpageObject, he selects it in the tree and drags it to the desired position within the webpage. However, not every node of the tree is placeable. To simplify the creation of a website, only nodes directly under the root node can be placed. If such a node is a CMContainer, its children are placed automatically on a default position within the WebpageContainer. This functionality greatly reduces the time used to create a website.

Although Drag&Drop is a feature used in many graphical user interfaces, it is still quite complicated to implement. The main problem is that often the Drag&Drop operation passes data from one process to another. This means that information must be transferred from one address space to another. This is very complicated without the support from the Operating System and a good API. Our case is a bit easier, the drag source (the structure tree) lies within the same process as the drop target (the JPanel representing the webpage). Nevertheless Drag&Drop remains difficult.

Fortunately, Java provides a Drag&Drop API to simplify the task. The package java.awt.dnd contains a few classes which facilitate Drag&Drop. The base principle is to distinguish between the source of the data, where the drag starts and the target of the data, where the drop occurs. An implementation of the DragSourceListener interface must be added to the drag source. This class observes if the mouse enters the area of the data enabled for Drag&Drop. It then changes the mouse cursor to give feedback to the user. To be able to recognize a drag operation, the drag source needs a DragGestureListener, this simple listener just tracks the mouse movements to recognize when the user wants to start a drag operation. If this happens it prepares the data to be transferred. To transfer data, a Transferable object must be created. This is the most complicated part of the Drag&Drop API. The Transferable is the actual object that is passed between the drag source and the drop target. It encapsulates the data to be transferred in a format suitable to the exchange between drag source and drop target. The drop target needs an implementation of the DropTargetListener interface to be able to receive the data. To enable Drag&Drop the graphical template editor implements all these interfaces in the package ch.ethz.globis.cmserver.gui.layout.dnd which contains four classes:

TreeDragSourceListener
Simply changes the mouse cursor when the mouse moves over a node suitable for a drag operation. This indicates to the user that this node can be dragged to place its WebpageObject on the screen.

TreeDragGestureListener
This class recognizes a drag gesture. When such a gesture occurs, it gets the currently se-
lected node from the tree and finds the name of the corresponding WebpageObject. Then it creates the TransferableWebpageContainer object to transfer the data and passes the found name to it.

**TransferableWebpageContainer**
This class implements the Transferable interface. It is needed to transfer the WebpageObject to the webpage. As the transfer happens within the same process, this class just passes the name of the WebpageObject to the webpage. The Webpage object then finds the corresponding WebpageObject in the collection of its children!

**WebpageDropTargetListener**
While the drag operation is still in process, this class shows a rectangle with the size of the WebpageObject to help the user to find a suitable place for the object. When the drop action occurs, it checks if the target is valid (no intersection etc.), then the TransferableWebpageContainer passes the name of the WebpageObject to the WebpageModel. The WebpageModel looks for the corresponding WebpageObject and places it on the desired location.

For more details on the package dnd refer to the API documentation in appendix C.

### 5.5.2 Editing the Webpage

Editing the webpage is quite easy to implement because the support of the java Swing architecture is large. Note that every object placed within the WebpageObject is either a WebpageComponent, this means a subclass of JLabel or a WebpageContainer which is a subclass of JPanel. Therefore, most of the work necessary is done by Swing. The graphical template editor must allow the user to do two different tasks. First, moving and resizing the object. Second, editing the properties of the object.

**Moving and Resizing WebpageObjects** Specialized mouse event listeners were implemented for WebpageComponent and WebpageContainer to support moving and resizing. When the mouse is moved over a WebpageObject, the appropriate listener takes care of the actions the user invokes. The classes WebpageComponentMouseHandler and WebpageComponentMouseMotionHandler are associated with every instance of WebpageComponent. The corresponding classes of WebpageContainer are WebpageContainerMouseHandler and WebpageContainerMouseMotionHandler. In the following a short description of these four classes found in the package ch.ethz.globis.cmserver.gui.layout.webpage. They are also shown in the UML diagram of the Webpage package in figure 5.3.

**WebpageComponentMouseHandler**
This simple class extends the MouseAdapter from the package java.awt.event. It is invoked when a mouse event like clicking or right-clicking occurs over the associated WebpageComponent. The class takes the needed actions to show the popup menu or to select the WebpageObject. In the case of a selection, the WebpageModel is called to inform the structure tree of the selection and to show the correct property toolbar. See the next section for information on the property toolbar.

**WebpageComponentMouseMotionHandler**
This class extends the MouseMotionAdapter from the package java.awt.event. It reacts on
5.5. HANDLING USER INPUT

every event type related to the motion of the mouse on the associated WebpageComponent.
When the user moves over the border of the object, this class changes the mouse cursor to
indicate that the WebpageComponent can be resized. In order to do this, the location of
the mouse cursor is determined, then the appropriate mouse cursor is shown. For example:
For a mouse location on the top edge of the component a cursor with an up and a down
arrow is shown.
When the user grabs the border of the WebpageComponent (moving over it and pressing
the mouse button) the WebpageComponent can be resized by dragging the border with
the mouse. In order to do this, the WebpageComponentMouseMotionHandler determines
the edge on which the resize operation started, then it calculates the new size of the Web-
pageComponent and calls the method drawContent to adjust the size immediately. If the
new size is too small or if it leads to an intersection, the background colour of the Web-
pageComponent is changed to red to indicate a forbidden size! When the user stops the
resizing with a forbidden size, the old size is restored.
To move a component, the user moves the mouse cursors into the WebpageComponent
and then presses the mouse button. While the button is pressed, the WebpageObject can
be moved around its WebpageContainer. The WebpageComponentMouseMotionHandler
determines the movement of the mouse since the button was pressed. This leads to a vec-
tor representing the displacement of the new location relative to the old location. With this
vector the new location can be calculated and the WebpageComponent is redrawn at this
new position. Again, if the new location is forbidden the background colour is switched to
red.

WebpageContainerMouseHandler Another simple class extending the MouseAdapter
from the package java.awt.event. It has the same function as the WebpageComponent-
MouseHandler only for WebpageContainers instead of WebpageComponents. In addition
it shows an extended context menu when the user right clicks on the container. In this
menu, one can select wether or not a grid to assist the positioning of components within
the container should be displayed.

WebpageContainerMouseMotionHandler
Like WebpageComponentMouseMotionHandler this class extends the MouseMotionAdap-
ter. It provides the same function for the WebpageContainer as WebpageComponent-
MouseMotionHandler has for the WebpageComponent. But it is a bit more complicated to
understand because the moving and especially resizing of a WebpageContainer requires a
very careful handling of the positions of its children. For example, assume that a Webpage-
Container containing a few WebpageTextComponents should be resized. The user starts
to resize the container by dragging its left edge to the right. This means that the container
gets smaller, therefore the offsets of its children get also smaller. The left hand picture of
figure 5.4 illustrates this situation. Then, as shown in the right-hand picture of figure 5.4,
the user releases the mouse in a forbidden position. The old size of the WebpageContainer
must be restored. This includes the old offsets which must be restored accordingly. The
WebpageContainerMouseMotionHandler takes care of this situation by calling the method
restoreChildPosition from WebpageContainer when a resize operation fails! As we
see, with a WebpageContainer few problems arise a which are not present with a Webpage-
Component.

For further details on the mouse event handler classes, please have a look at the API of
the package webpage in appendix C.
Editing Properties of WebpageObjects  We have seen how a WebpageObject is moved around the webpage. Next we discuss how its properties can be changed. Every WebpageObject has a property toolbar associated. When the object is selected, the WebpageModel gets the associated toolbar from the WebpageObject and passes it to the XSLGUIPanel which shows it in its top right panel. Depending on the type of WebpageObject the property toolbar contains comboboxes and buttons to edit the properties of the WebpageObject. For example, the only editable property of a WebpageCollectionContainer is the background colour. Therefore the property toolbar of the WebpageCollectionContainer contains only one button to change the background colour. On the other extreme, the WebpageMixedContainer has a background colour property, furthermore its mixed text’s font, font size, font style font colour and background colour can be changed as well. Therefore, the property toolbar of the WebpageMixedContainer contains a button to change the background colour of the Container, in addition it also offers the possibility to edit the properties of the mixed text: There are two comboboxes to change the font family and the font size, one button to edit the font colour and another button for the background colour of the mixed text. Again it can be seen that although there are some similarities between the different toolbars, their appearance differs and depends on the concrete type of the WebpageContainers whose properties it manages. It is a straightforward idea to define an interface PropertyToolbar, which every toolbar associated with a WebpageObject must implement. Additionally the toolbar must extend JPanel in order to be placeable within the XSLGUIPanel. Following are the three methods defined by PropertyToolbar:
5.5. HANDLING USER INPUT

public void setWebpageObject(WebpageObject webpageObject)

Sets the WebpageObject for which the toolbar is responsible.

public WebpageObject getWebpageObject()

Gets the WebpageObject for which the toolbar is responsible.

public void setVisible(boolean visible)

This method is also defined by JPanel. By adding it to this interface, it is enforced that the programmer should override the setVisible method of JPanel. The method must be called when the property toolbar is shown or hidden. When it is shown, it is the programmers responsibility to adjust the toolbar reflecting the latest changes of the properties of its WebpageObject, e.g. when the WebpageObjects background colour is changed, the toolbar should be adjusted to the change before it is made visible again.

The interface PropertyToolbar and all its implementations are found in the package ch.ethz.globis.cmserver.gui.layout.properties. To give an impression how such an implementation of PropertyToolbar looks like, we inspect the class CMTextPropertyToolbar which manages the properties of WebpageTextComponent. An image of CMTextPropertyToolbar is shown in figure 5.5. Below, we discuss some of its methods.

![CMTextPropertyToolbar](image)

Figure 5.5: The CMTextPropertyToolbar

private void fillCombobox()

Fills the comboboxes with the available fonts and the font size.

private void fontChanged()

This method is called when the user changes the font. It sets the new font in the WebpageTextComponent and updates the comboboxes.

public void setVisible(boolean visible)

This method overrides the setVisible method of JPanel. It does nothing more than call updateToolbar and then invokes the superclass with super.setVisible(visible).

private void updateToolbar()
This method must be called to adjust the toolbar to the latest changes of the properties of its WebpageObject. The method is quite important because some properties, e.g. the background colour can be changed when the corresponding property of the WebpageContainer this WebpageTextComponent belongs to is modified.

For more details on the package properties have a look at the API documentation in appendix C.

Here we conclude the discussion of the implementation of the frontend. The next chapter will show how the backend is implemented and how the creation of the templates work.
Chapter 6

Implementation of the Backend

This chapter describes the architecture of the backend. The backend is invoked when the user presses the save button, it then starts the creation of the XSL templates. As already mentioned, this process works in two steps. First an XML representation of the layout of every WebpageObject is created, then these XML files are transformed into the correct templates. We will adopt this division in our documentation as well. First we describe how the XML representation of the layout is created. Then we explain the transformation to XSL templates.

6.1 Creating an XML Representation of a Page

All classes needed for the backend are found in the small package ch.ethz.globis.cmserver.gui.layout.stylesheet. It only contains two classes: The class XMLLayoutGenerator creates the XML layout representation. The class XMLLayoutLoader does the opposite, it takes an XML representation and loads the positions and other properties of the corresponding WebpageObjects. The XMLLayoutLoader is discussed later in this chapter. For the moment we will focus on the XMLLayoutGenerator.

A short comment before we start, the Java library used to create and manipulate XML files within the CMServer is the popular jdom library [13]. Accordingly, the graphical template editor also uses jdom!

The job of the layout generator is to create XML files containing all information about the desired layout of the webpage. Similar to the templates, each of the generated XML files will be associated with a single CMComponent whose layout it represents. To incorporate this change into the database, we must adjust the data model of the CMServer. Fortunately, only the CMTemplate data type must be modified. The new definition is shown below. Compared with the old implementation, it contains a new text attribute layoutTemplate to store the XML layout description and a new method getLayoutTemplate to retrieve the layout template from the database.
CHAPTER 6. IMPLEMENTATION OF THE BACKEND

```typescript
type cmtemplate {
    name : string;
    layoutTemplate : text;
    getLayoutTemplate : () -> (layoutTemplate: string);
    getXSLTemplate : () -> (template: string);
}
```

After having presented how the new XML layout representation is stored. We will have a look at its format in the next section.

### 6.2 The DTD of the XML Layout Representation

The format of XML files can be represented best by the use of the data type definition DTD [12]. In the following we will explain how the DTD for XML layout files is structured. To find a suitable DTD we must have a look at the way webpages are represented in HTML. In HTML complex page layouts are created with nested tables. This means, a webpage is represented by a table whose cells contain other tables or basic content such as texts, links or pictures. Figure 6.1 shows a typical table layout as it is visualized by the Microsoft FrontPage HTML editor. It is certainly a good idea to create a XML representation that makes it easy to create such a table layout. Furthermore the DTD must contain all information needed to create the correct fonts and colours within the webpage. The most easy way to explain the format is to go through the different elements step by step. The entire DTD can be found in appendix D.

![Figure 6.1: Table layout as visualized by Front Page](image-url)
6.2. THE DTD OF THE XML LAYOUT REPRESENTATION

Webpage  The Webpage object is represented by the element webpage. It contains a layout element and the fontlist which contains a list of the fonts used within the webpage. The layout element is the base of the table layout. A layout can contain a number of rows. Each row has a number of columns. The columns represent the cells within the table. A column can contain other elements like collection, mixed, text etc. which we will explain later on. The column has two attributes rowspan and colspan. Normally these attributes have the value 1. But if an element within the column is larger than the size of a cell, it uses the space of the columns in the neighborhood. The number of columns overlapped horizontally is indicated by colspan, the vertically overlapped columns by rowspan. This concept is very similar to the way HTML table layout works, but it is flexible enough to be used for the creation of other markup languages as well. The attributes store the properties of the webpage. The name and the oid attribute are taken from the CMContainer represented by the WebpageObject. They are used to be able to identify the element and to link it correctly with its children.

```xml
<!ELEMENT webpage (layout,fontlist)> <!ATTLIST webpage
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  fixedWidth CDATA #IMPLIED>
```

The layout contains multiple rows. A layout element is part of every element representing a CMContainer.

```xml
<!ELEMENT layout (row+)> <!ATTLIST layout
  width CDATA #REQUIRED
  height CDATA #REQUIRED>
```

Rows simply contain column elements.

```xml
<!ELEMENT row (column+)> <!ATTLIST row
  width CDATA #IMPLIED>
```

The column element contains either one of the other elements or it is an empty column with just the special spacer element.

```xml
<!ELEMENT column (spacer|collection|mixed|text|mixedText
  |link|picture|navigation|sitemap
  |defaultcomponent|defaultcontainer)> <!ATTLIST column
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  colspan CDATA #REQUIRED
  rowspan CDATA #REQUIRED
```

The font elements are created by the different WebpageComponents during the creation of the XML layout. The XMLLayoutGenerator collects all font elements to create the fontlist element for the webpage. This font list is used to generate a cascading stylesheet (CSS) with all fonts needed in the webpage.

```xml
<!ELEMENT fontlist (font*)>

<!ELEMENT font EMPTY> <!ATTLIST font
  font_id CDATA #REQUIRED
  name CDATA #REQUIRED
  style (plain|bold|italic|bolditalic) #REQUIRED
  size CDATA #REQUIRED
  color CDATA #REQUIRED>
```

**WebpageCollectionContainer**  CMCollections are subtypes of CMContainer, therefore like the webpage they contain a layout element to structure the appearance of their children. The attributes give informations on the properties of the collection. To be able to restore the WebpageCollectionContainer, the size and the coordinates on the screen are also stored within the attributes. As in the webpage, the name and the oid attribute are used to be able to identify the element and to link it correctly with its children and with the parent element.

```xml
<!ELEMENT collection (layout)> <!ATTLIST collection
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>
```

**WebpageMixedContainer**  The mixed container is very similar to the collection. It also has the same attributes. The only difference is the mixed text.

```xml
<!ELEMENT mixed (mixedText,layout)> <!ATTLIST mixed
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>
```

The properties of the mixed text are represented by the element mixedText.
6.2. THE DTD OF THE XML LAYOUT REPRESENTATION

```xml
<!ELEMENT mixedText (font?)> <!ATTLIST mixedText
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED
>
```

**WebpageTextComponent**  The layout of CMText content is represented by the element `text`. It has no special attributes. Its only child is the `font` element which contains the font, font style and font colour used by the text. If the text belongs to a link, the attributes for size and location are not necessary because they are already stored in the link element.

```xml
<!ELEMENT text (font?)> <!ATTLIST text
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
>
```

**WebpageTextLinkComponent and WebpagePictureLinkComponent**  Links are a bit more complicated than texts. In HTML, links have an URL and a resource. Normally, the resource is a text or a picture. This is also the way the graphical template editor supports links at the moment. However, in theory nearly every component of a webpage could be a link. Therefore the `link` element can contain any of the other elements as a child. The link elements has the usual attributes.

```xml
<!ELEMENT link
  (collection|mixed|text|link|picture|defaultcomponent|defaultcontainer)> <!ATTLIST link
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED
>
```
WebpagePictureComponent  The layout of a CMPicture is stored in the picture element. It is a very simple element with just the usual attributes to be able to identify and restore the component and no children. The attributes for size and position are only implied because if the picture belongs to a link they are not necessary.

```xml
<!ELEMENT picture EMPTY> <!ATTLIST picture
    name CDATA #REQUIRED
    oid CDATA #REQUIRED
    parent_id CDATA #REQUIRED
    backgroundColor CDATA #REQUIRED
    width CDATA #IMPLIED
    height CDATA #IMPLIED
    x CDATA #IMPLIED
    y CDATA #IMPLIED
>
```

WebpageNavigationComponent  Navigations are represented by the navigation element.

```xml
<!ELEMENT navigation (font?)> <!ATTLIST navigation
    name CDATA #REQUIRED
    oid CDATA #REQUIRED
    parent_id CDATA #REQUIRED
    backgroundColor CDATA #REQUIRED
    width CDATA #REQUIRED
    height CDATA #REQUIRED
    x CDATA #REQUIRED
    y CDATA #REQUIRED
>
```

WebpageSitemapComponent  Sitemaps are represented by the sitemap element.

```xml
<!ELEMENT sitemap EMPTY> <!ATTLIST sitemap
    name CDATA #REQUIRED
    oid CDATA #REQUIRED
    parent_id CDATA #REQUIRED
    backgroundColor CDATA #REQUIRED
    width CDATA #REQUIRED
    height CDATA #REQUIRED
    x CDATA #REQUIRED
    y CDATA #REQUIRED
>
```

WebpageDefaultComponent  Default components also have a layout, e.g. the background colour can be set. In addition it is important to know the oid of the component and its parent. Default components are represented by the element defaultcomponent.

```xml

```
6.3. THE XMLLAYOUTGENERATOR

<!ELEMENT defaultcomponent EMPTY> <!ATTLIST defaultcomponent
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>

WebpageDefaultContainer  Default containers are represented by the defaultcontainer element.

<!ELEMENT defaultcontainer layout> <!ATTLIST defaultcontainer
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>

6.3 The XMLLayoutGenerator

As explained above, every CMComponent needs an associated XML file describing its layout. Therefore it is convenient to walk step by step through the structure of the webpage and to create the XML files for every child WebpageObject. This approach is also supported by introducing a method public Element createXMLLayout() into the interface WebpageObject. The method returns a jdom XML Element[12] describing the layout of the WebpageObject. Next, we analyse how the createXMLLayout() method can be implemented by WebpageComponent and WebpageContainer.

WebpageComponents just have to inspect their properties and set the attributes to these values. The WebpageTextComponent, WebpageTextLinkComponent and WebpageMixed-TextComponent also have to create the correct font elements. When this job is done, an XML element can be returned. Unfortunately, in the case of WebpageContainers the creation of the XML elements is not as simple as for WebpageComponents. Out of the children positions within the screen, WebpageContainers must create a table layout, and they are responsible for the correct size and position of their children as well. To do this, the XMLLayoutGenerator was developed. It is used within the method createXMLLayout and contains two inner classes to facilitate the creation of the XML layout element. The inner class Layout basically represents a table, the class Cell implements a table cell. The algorithm used to create the layout is quite complicated. In order to understand better the following explanation of the algorithm, refer to the example situation shown in figure 6.2.
CHAPTER 6. IMPLEMENTATION OF THE BACKEND

Figure 6.2: Creating the Table Layout

The Layout Algorithm

1. A jdom element which stands for the container is created. The properties of the container (background colour, size, position, oid, parent oid etc.) are extracted and the corresponding attributes are generated and added to the element.

2. Now the table is constructed. In order to do this, a new instance of the inner class Layout is generated. The Layout instance calls its method createAxis which it- erates through the children of the WebpageContainer and retrieves their locations. createAxis uses the points to generate a grid spanning the whole WebpageCon- tainer. The points are sorted top-down and from left to right.

3. The layout class now has constructed a grid of the table layout. It calls the method createCells which, As the names indicates, creates the cells of the table layout by iterating through the points of the grid. For every grid point, the list of Webpage-Objects is consulted to find an object with a matching location. If such a WebpageObject is found, it must be known if it spans multiple rows and columns. In order to do this, the grid points in the neighborhood of the current point are checked. If the WebpageObject overlaps these points, we have an object spanning multiple rows and columns. Having collected all this information, the algorithm can create a Cell object. The constructor of the Cell has parameters to assign the found WebpageObject and to set the correct cell- and rowspan. If no WebpageObject with the same coordinate as the current point is found, an empty spacer cell is created. Such a cell corresponds to the empty space between the different CMComponents on a webpage!

4. When the iteration through the grid points is finished, the Layout class has built a collection of cell objects ordered by their coordinates. Some of these cells are empty, some contain WebpageObjects. Now the method createXML of the class Layout is invoked to create the Element layout and to assign the needed attributes.
5. The collection of cells is iterated, and for every cell the method `createXML` is called. `createXML` creates an XML Element of the type `column`. If the cell has a `WebpageObject`, the corresponding `createXMLLayout` method which returns an XML Element is invoked. The resulting element is added to the `column` element. In addition, it is stored in the database and assigned to the `WebpageObject` by the call of `saveToDatabase`. At the end, the `column` element is returned.

6. The Layout class adds every `column` element to the `layout` element. When the iteration through the cells is finished we return the `layout` element and hence the layout of the container is complete.

### 6.4 Generation of XSL Templates

We have presented how the layout of every `WebpageObject` is generated. This section explains how this XML layout file is used to create the desired XSL template. For every type of `WebpageObject`, there is an XSL stylesheet that takes the XML layout for a `WebpageObject` as input and transforms it to an XSL template. The transformation is invoked by the method `transformXSLStylesheet` defined in the interface `WebpageObject`. As the method returns a `String` object, it is very easy to save the result in the database. The generation of the template is quite easy to understand. The XSL stylesheet contains templates matching the elements defined in the XML file. A template is invoked when the processor finds an element corresponding to it. The template then inspects the attributes of the XML element and uses them to create the new template. To get a better understanding of the whole process, we will discuss some of the XSL code. Every stylesheet used for subclasses of `WebpageContainer` contains the same template rule matching the `column` element. It is responsible to create the columns within the HTML table layout and it must generate the calls for the templates contained within the column. In the following the begin of this template which creates the column element is given.

```xml
<xsl:template match="column">
    <xsl:param name="parent_id">0</xsl:param>
    <td>
        <xsl:attribute name="colspan">
            <xsl:value-of select="@colspan"/>
        </xsl:attribute>
        <xsl:attribute name="rowspan">
            <xsl:value-of select="@rowspan"/>
        </xsl:attribute>
        <xsl:attribute name="width">
            <xsl:value-of select="@width"/>
        </xsl:attribute>
    </td>
</xsl:template>
```
In the resulting template the column definition created by the above template looks as follows:

```xml
<td colspan="2" rowspan="1" width="40%" height="5%" valign="top" bgcolor="#ff0033">
```

The greatest difficulty when generating the templates is the correct linking of parent CM-Containers with their children. In the CMServer, the stylesheet finally used to create the webpage is a composite of the templates of the CMComponents contained in the webpage. To make sure that these components are placed at the correct position, the templates must be called in the right order. This problem is solved as follows: As explained above, every stylesheet for a WebpageContainer contains the same template to match a column node within the XML layout representation. In the above example we presented a first excerpt from it, which generates the column elements with the colspan and rowspan values found in the attributes. In a second step, the column template takes the oid attribute of the container and the oid attribute of the current child of the column and creates the call to the child’s template. The code to do this is shown below. Note that it was necessary to use ```elements to create the call because the XSL processor was not able to transform templates that used two different namespaces for the generating and the generated XSL code.

```xml
<xsl:for-each select="child::*">
   <xsl:text disable-output-escaping="yes">
      &lt;xsl:apply-templates
         select="&quot;//webobject[@oid='"$
      </xsl:text>
   <xsl:value-of select="\$parent_id"/>
   <xsl:text disable-output-escaping="yes">
      '/
   </xsl:text>
   <xsl:apply-templates
      select="&quot;//webobject[@oid='"$
   </xsl:text>
   <xsl:value-of select="attribute::oid"/>
   <xsl:text disable-output-escaping="yes">
      '/
   </xsl:text>
</xsl:for-each>
```
The above snippet generates the following template call within the XSL template of a CMContainer. Below, we show the call and the table column in which it is embedded.

```xml
<td colspan="2" rowspan="1" width="40\%" height="5\%" valign="top" bgcolor="#ff0033">
  <xsl:apply-templates select="//webobject[@oid='OM_2112']/components/webobject[@oid='OM_2271']"/>
</td>
```

Of course, the child must have the correct matching clause to be called. The following code snippet shows the first part of the template matching a text element. Note, how the matching clause is generated with the parents oid and the oid of the text-element itself.

```xml
<!--This is the template snipplet responsible to create the correct rendering of a cmtext.-->
<xsl:template match="text">
  <xsl:text disable-output-escaping="yes">
    <xsl:template match="//webobject[@oid='
      <xsl:value-of select="@parent_id"/>
    ']/webobject[@oid='
      <xsl:value-of select="@oid"/>
    ']">
      .
    </xsl:template>
  </xsl:text>
</xsl:template>

The above XSL code leads to the following header in the resulting XSL template for a CMText, which uses the oid values of the parent container and the text component to ensure that the template for the correct CMText webobject is called.

```xml
<xsl:template match=""//webobject[@oid='OM_2112']//webobject[@oid='OM_2271']"
>
```

This concludes our presentation of the basic mechanisms at work in generating templates. The code of all templates involved can be found in appendix D. There is however one issue that has not been discussed yet. One has to decide where these XSL stylesheets are stored. Clearly it is not desirable to hardwire them into the XCM Sitemanager application. A possible solution would be to store them in the database as well. But in the implementation of the graphical template editor, we have chose a different approach. The stylesheets are part of the XCM Sitemanager application, they should be
independent from any database. Therefore, they are distributed with the XCM Sitemanager
and can be stored in a specific directory within the filesystem. Of course the application
must know the actual path to these stylesheets. In the Windows Preferences menu there is a
dialog to set the preferences for the XCM Sitemanager. Within this dialog one can set the
path to the stylesheets. The stylesheets must all lie in the same directory.

6.5 Loading Existing Templates

Of course, it should be possible to load a before-hand defined layout of a webpage from
the database. The process to do so is simple. First, as if a new layout would be created,
an empty layout is generated. Note, that this empty layout contains already all Webpage-
Objects, however they are not yet placed and therefore invisible. When the empty layout
is loaded, the XML file for the CMContainer representing the layout is extracted from the
database. This file contains the layout for all CMComponents contained within the data-
base. The class XMLLayoutLoader parses the XML file and sets the correct properties (e.g.
colour, size, position, font) for the different WebpageObjects within the webpage. That is
the reason why all the information about size and position on the screen is also saved in
the XML file. Now, the WebpageObjects are displayed at the correct position with appro-
priate size, font and background colour. By invoking setPlaced(true) they are
made visible and the load-process finishes. We will not go into more details because the
implementation is straightforward and not very interesting. For further information refer
the API documentation in appendix C.

6.6 Extending the Editor

In the last section of this chapter we show how the graphical template editor can be ex-
tended to support new data types. The extension is done in five steps.

1. The first task should be to consider what kind of layout properties the new data type
has. Then the DTD of the XML Layout representation must be extended with new
elements to be able to represent the new type and its properties.

2. Depending on the base type of the new data type, a new subclass of WebpageCom-
ponent or WebpageContainer must be created. For a CMContainer the new class
extends WebpageContainer, for any other CMComponent WebpageComponent is ex-
tended. The new class must be able to represent all new layout properties defined in
step 1 on the screen. Consequently, the createXMLLayout method should produce
the new XML elements as defined above.

3. Create a subclass of PropertyToolbar to modify the properties of the new class.

4. Adjust the factory class WebpageObjectFactory to be able to create instances of the
new class. For WebpageContainers, the class XMLLayoutGenerator should also be
consulted and checked if the new type requires an adjustment.

5. The last step is to create an XSL stylesheet which creates the XSL template from the
XML layout representation of the new data type.
As can be seen, extending the graphical template editor is quite easy. In the factory class only must the programmer change existing code to be able to add and create the new type. This ensures that the graphical template editor should scale well in the future.
Chapter 7

Conclusion

This chapter concludes this diploma thesis. First, we will analyse if all the objectives of the thesis have been met. This is followed by a critical review of the architecture and an enumeration of its shortcomings. At the end of this chapter, we look into the future and try to give some hints for possible enhancements and extensions of the system.

7.1 Fulfillment of Requirements

The requirements, as presented in the first part of this report and described in chapter 4 were fulfilled. The following is a list of the requirements and how they were implemented.

**The complexity of XSL must be hidden** The creation of XSL is totally hidden from the user, as the editor generates all templates automatically. It is however possible to edit the templates manually.

**Use the Metadata of the Data model** This point was the crucial requirement for the success of this thesis. As explained in chapter 5 and 6, the structure of the application reflects the data model of the CMServer. This is very good visible in the UML diagram in figure 5.3.

**Support for every Base Data Type** The graphical template provides an XML layout representation and a corresponding stylesheet for every base data type (see appendix D).

**Handling of unknown Data Types** Unknown base types can be incorporated into webpages created with the graphical template editor. They have a default layout (see appendix D). But if necessary, the editor can be extended to handle these objects as if they were built-in types.

**Extensible Architecture** The architecture is very easy to extend. Especially new data types are integrated quite easily as we have shown in chapter 6.

**Editing and Positioning of CMComponents** CMComponents can be placed and edited by simple mouse operations. See the tutorial (appendix B) and chapter 5 on the frontend for details.
Smooth Integration into the XCM Sitemanager There is certainly no break between the graphical template editor and XCM Sitemanager. We have used the same tree view and the icons fit very good into the application. The use of our application is also similar to the already existing part of XCM Sitemanager.

7.2 A Critical Review of the Implementation

The implementation of the graphical template editor certainly has been a success, since basic architecture of the system was well designed. Especially the representation of the data types on the screen and the division of the creation process of the into two steps has proven to be appropriate. This ensures that the system is easy to extend and also simple to understand. Nevertheless there are some points to critic. Although the base concepts are easy to understand, sometimes the code to implement them can still be better. The reason for this is, that the development process of the graphical template editor was not adequate from time to time. Especially the handling of user interaction has not been fully specified in the beginning. Also, we are not fully satisfied with the implementation of the model view principle in the editor. In retrospection, we have to admit that a further separation of model and view would be possible. The solution would be to define two interfaces WebpageObjectView and WebpageObjectModel instead of the single interface WebpageObject. Methods like createXML, which are typical for the model of WebpageObjects, could then be separated better from the methods used only to manage the appearance of objects on the screen. Another critical point is event handling. The mouse listeners could be implemented more efficiently. May be, it is possible to use the interface WebpageObjects and to write a single event-class suitable for all its subtypes.

7.3 Future Enhancements and Possible Extensions

As seen above, there are a few things one could correct in the graphical template editor first.

- Review the code and look for methods that could solve their tasks more easily.
- Improve the separation of model and view by a better implementation of the model-view principle.
- Improve the event handling.

But there are also some further enhancements possible. These are just things that were not implemented because the time for this thesis was limited.

- Support for more data types. For example, one could program classes for all the types used in the globisweb database.
- Allow the user to assign borders to WebpageObjects. In modern webpages, borders can play an important role.
- Create a better navigation template. The currently used navigation template is suitable, but it could be more fancy.
7.3. FUTURE ENHANCEMENTS AND POSSIBLE EXTENSIONS

- Sometimes webpages contain images whose different parts contain different links. Support for such position-sensitive links on images would be a very interesting feature.

- Allow the definition of lists out of CMCollections.

- Background images for CMContainers would also be an interesting feature.

- Maybe it is even possible to support forms and javascript?

There is still a lot to do. We hope that the graphical template has been implemented well enough for other people to take the time to do these corrections and improvements!
Appendix A

Setup of XCM Sitemanager

This appendix describes how the new, extended version of XCM Sitemanager is installed. The installation includes the setup of a CMServer, as it is convenient to quickly test the created websites. A CD-ROM with the binaries and libraries needed to run XCM Sitemanager and the CMServer was created as a part of this diploma thesis. It also contains the API documentation, a sample database for the tutorial in the next section, this report including all images and last but not least an adapted version of the globisweb database. In this appendix we explain the setup based on this CD-ROM. But we will also specify the software needed to do the installation without it.

A.1 Installation

The Installation is quite easy.

1. A Java Development Kit (J2SE v1.4.x) environment is found on the CD-ROM in the java directory. Execute the setup file to install java.

2. By clicking on the setup file, install the Apache Tomcat application server (version 4.1.18) found in the tomcat directory on the CD-ROM.

3. Copy the folder cmserver from the CD-ROM into the webapps directory of your Tomcat installation.

Of course, if you already have a Java Development Kit and a Tomcat application server installed, you do not need to perform the first two steps.

A.2 Starting and Configuring XCM Sitemanager

To start XCM Sitemanager, a batch file XCMSiteManager.bat was written. It is found at $tomcat_home$\webapps\cmserver\WEB-INF\. Double click the batch file to XCM Sitemanager. The first time the XCM Sitemanager is started, you must set a few properties.
APPENDIX A. SETUP OF XCM SITEMANAGER

Figure A.1: The Preferences Dialog

For this purpose, open the dialog Windows-Preferences shown in figure A.1. The property-fields for the language settings and for the output type should be self-explaining. The field URL of spacer.gif must point to the address of the webserver, which holds the spacer.gif used within empty cells of HTML tables. The field Path to xsl-files should contain the path to the location where you save your stylesheets used to create the XSL templates. In the default installation this folder is: $tomcat\webapps\cmserver\WEB-INF\stylesheets.
The field Path to images must point to the folder with the images used within the webpage. It's very important not to forget this. Often, a bad or missing setting of this path is the reason for a malfunction of the graphical template editor.

A.3 Required Software

As a reference and in case you want to install XCM Sitemanager without the CD-ROM, in the following a detailed list of the required software.


2. The Java version of the Xalan XSL processor (version 2.4). Available at xml.apache.org/xalan-j/index.html.

3. The jdom library for XML (version 1.0 beta 8). The library is found under http://www.jdom.org.

4. The OMS Java database (version 2.0 release 47), unfortunately the tutorial works only with this version. Downloadable at http://galbraith.inf.ethz.ch:8080/omsjava/download.

5. The sigtec libraries, found at http://gordon.inf.ethz.ch/sigtec.
6. The classes for XCM-Sitemanager and the CMServer found on the CD-ROM.
APPENDIX A. SETUP OF XCM SITEMANAGER
Appendix B

A Short Tutorial on the Graphical Template Editor

In the following tutorial we explain the use of the graphical template editor. Users which are not yet familiar with XCM Sitemanager should consult [3] first, because this tutorial assumes knowledge of XCM Sitemanager. The cmserver\db\tutorial.dump on the CD-ROM was especially created for this tutorial. All the example below refer to this database.

B.1 Configuring XCM Sitemanager

In order to make sure that this tutorial works, we must first check if all the preferences of XCM Sitemanager are set correctly. The sample database is the website of the small swiss handball club “STV Luzern” and it is in german. Therefore, we first check if there is already german language setting. To do so, open the dialog Site-Preferences (figure B.1) and make sure that the combobox with all languages contains a german language setting. If there is none, create a new german language setting. Use the same parameters as shown in figure B.1. Next, we need to check if all other properties are set correctly. Open the

![Figure B.1: The Languages Dialog](image)

Windows-Preferences (shown in A.1). In this dialog, check if the languages are set to german.
Then make sure that the images-path points to the directory containing the images for the example database. The images for the tutorial and for the CMServer reference database globisweb [2] are found in the folder images. To display the created webpages correctly on the screen, a transparent spacer image must be provided by a webserver. Check if the path to the spacer-file is valid. If you installed XCM Sitemanager as described in appendix A, the path-entry is: http://localhost:8080/cmserver/web/pics/spacer.gif. The last path-entry you should verify is the folder with the stylesheets needed to create the templates. The stylesheets are found on the CD-ROM in the folder cmserver\WEB-INF\stylesheets\stylesheets\. Of course, the desired markup must be set to HTMLTemplate. If all settings are correct, open the tutorial.dump file with Site-Open. On the CD-ROM the file is found in the directory cmserver\db\. Have a look at the structure of the site and inspect the different data types. If you think you are familiar with the site, you can start with the tutorial.

### B.2 The Layout GUI Tab

To use the graphical template editor, chose the tab labelled with GUI Layout. A panel as shown in B.2 appears. The tree on the left side shows the structure of the website. Take some time to navigate through the tree and inspect how the different types of CMComponents are represented. Page nodes are identified by the .html ending of their names and by the white page-icon with a globe. Chose one of the page nodes in the structure tree. Observe the buttons above the tree. Their function is explained in figure B.3. As soon as you have selected a page, the load-button is activated. This happens when there is already a layout defined for the selected node. With the load button you can bring it to the screen. The create button is enabled as well, with this button, the the creation of a new layout is initiated.
B.3 Loading an Existing Layout

As we said, the database already contains sample layouts for all pages within the website. To start the tutorial, we will load one of these samples and have a look at its structure. Select the index.html page of the root node, then click the load-button. The program now loads the layout for this page. When its done, you should see the page as shown in figure B.5. Have a look at the different CMComponents on the webpage. As you see, there are two CMPictures, a CMText, a CMLink and a large (empty) CMNavigation. Each component is surrounded by a black border. You can choose a component by clicking into it or by selecting its node in the tree. Now select a component, to indicate the selection, a semi-transparent red shade appears. As each component has other properties to modify, the buttons and comboboxes in the panel above the page change depending on the chosen component. Experiment a little with the selection of the different components to see what properties you can modify. In figure B.4, all possible tools are shown and explained. The next section

B.4 Editing a Component

Now we will change the layout of the index.html page. To begin, we change the background colour of the page. In order to do this you must first choose the page node in the tree. The tool panel above the page changes and shows a single button with the letter “B” (for Background). Click the button, in the dialog that appears, select the colour you like, then click OK. Before the colour changes, you are asked if you also wish to modify the colour of the children. If so, choose yes. Note how the colour has changed, also notice that the background of the (B)ackground button was adapted. Change the colours of some of the
others component as well. The next step is to modify the font of the large text in the middle of the webpage. Its name is Willkommenstext. On selecting this text, the tool panel changes, showing a lot of different layout options. Choose a new font for the text, and note how the text changes. Experiment a little with the tools in the panel. Try to change the font to bold or italic (or both) and change the text colour.

B.5 Moving a Component

Now we have a look at the CMLink called IHV Link. Notice that you could change its font and colour as well. But for now we want to do something else: It would be nicer to have the link beside the logo on top of the page. To do so, move the mouse into the link component, press the button down. As long as you press the button, the component can be positioned by dragging it with the mouse. When the link moves over another component or is too close to the edge of the page, it becomes totally red to indicate a forbidden position. Move the link to the top of the page beside the logo, release the mouse button. See how it is placed at its new position. Try moving the other components. When you release the mouse in a forbidden position, the component is moved back to its old location. Also notice that the webpage is enlarged automatically when you move a component over its bottom edge.
B.6 Resizing a Component

Of course it is also possible to change the size of a component. This is done by moving
the mouse over the edge you want to resize. When the mouse cursor changes to an arrow,
press the mouse button and hold it down. As long the button is down, you can modify
the size of the component. Again, if the component becomes too large or too small or if it
intersects with another component, it becomes red to indicate a forbidden size.

B.7 Showing the Grid

Within their container, components are aligned to a fine grid with a width of 10 pixels. You
can show that grid by right-clicking into an empty space of the container. In the menu that
appears select Show Grid. To remove it right-click again and select Hide Grid. Alternatively,
select the container in the tree, then right-click on it. The same menu to show the grid pops
up.

B.8 Save or Cancel a Layout

We have now gone through have the basics of editing a website. Take some time get
a little experience. When your are done, there are two possibilities. If you like your
new layout, hit the save-button to store it in the database. The webpage is automati-
cally closed at this moment. Please note that you lose the old layout when you save
a new one. To restore an old layout, you must manually reassign the templates in the
normal layout panel. If you do not want to keep the new layout, just click the cancel-
button to cancel the editing. The webpage is closed as well. When you saved the layout,
maybe you want to have a look at how it is rendered in HTML. If your CMServer is al-
ready running and configured correctly, just open the URL of the page in the browser.
We will not show the setup of the CMServer here, for informations on this refer to [2].
However, if you have the default installation of XCM Sitemanager as described in the
chapter before, your CMServer should work (if Tomcat runs), and the page is reachable
you must have the correct database in the \textit{db} directory of the CMServer. Otherwise you
will not see anything.

B.9 Creating a New Webpage Layout

In this section, we learn how a new layout is created. To do so, open the \texttt{Gegner} folder,
then select the page node \texttt{index.html} and click the create button (figure B.3). A dialog shows
up. Enter the desired name of the new layout and specify whether the page has a fixed
width or if it should be relative to the width of the browser window. If you choose a fixed
width, you must also specify its value. When you are satisfied, hit the OK-button. An
empty page as shown in B.6 appears. Note, that all the nodes in the tree are red. This
means they have not yet been placed. The next section shows how to do that.
APPENDIX B. A SHORT TUTORIAL ON THE GRAPHICAL TEMPLATE EDITOR

Figure B.6: Creating a new Layout

B.10 Placing Components

Select the component called *logo*. Press the mouse button and drag the logo node into the page. As soon as you are within the page, a green rectangle represents the current position of the logo. Move it around the page. Release the mouse at the location where the logo should be. The STV logo appears and its node in the tree turns green because now it is on the page.

Now, we want to place a CMContainer. Select the container node called *Gegnerliste 1*. Drag it into the page like the logo before. Notice that the size of the rectangle representing the container changes while it is dragged around. This is because the size of the CMTexts within the container varies depending on the position. Again, if a location is illegal, the container turns red. Release the container at the desired location. The components of the container are drawn as well. The reason for this is, that it is only possible to drag components into the webpage which are directly below the page-node. All other components are drawn automatically when its parent container is placed on the webpage. Try to move one of the CMText components contained within the other container (Gegnerliste 2) to the page. It is not possible. Place the Gegnerliste 2 on the page. Its children are placed automatically
B.11 Working with Containers

Now we can test if the containers work correctly. Try to resize and move a container and its children. As you see, it is neither possible to move a component out of its container nor to make the container too small for its children. This is because such actions are illegal and should therefore be avoided. It is also possible to show an aligning grid for a container, just right-click and select *Show Grid*.

B.12 Removing Components

To remove a component right-click and select *Remove Component*. The component will disappear from the webpage and its node in the tree turns red again. Of course you can place the component on the page again. Note, that components removed from a container can only be placed again, if the whole component is removed and placed again.

B.13 CMNavigations and CMSitemaps

Last but not least, we want to place the CMNavigation. CMNavigation and CMSitemap are special components. Their content is generated on demand when the page is rendered. Therefore we cannot show it in the graphical template editor. Because of this, they contain a placeholder text only. Nevertheless, they can be placed and their properties can be edited like any other component. Try to place the CMNavigation and change its background colour.

B.14 Hints

As you see, using the graphical template editor is easy. Every possible action has been mentioned in the above examples. Do some experiments to become more familiar with the user interface. Carefully examine all pages and their sample layouts in the tutorial database. Especially, the page *index.html* in the directory *Spielberichte* is interesting because a CMMixed is used there. If you know how to create a website with the editor, go and have a look at the created templates. You can find them in the normal layout editor of XCM Sitemanager. On the CD-ROM you also find a version of the globisweb database. Load this database to have look how the graphical template editor treats unknown CMComponent types!
Appendix C

The API of the Graphical Template Editor

C.1 Introduction

This appendix covers the API of the graphical template editor. Every class is discussed with all of its methods. The classpath of the package layout is ch.ethz.globis.cmserver.gui. The name of the layout package is omitted in the class-names.

C.2 layout

C.2.1 layout.DesignGUITree

\begin{verbatim}
public DesignGUITree
extends JTree

Represents the structure of a page in the website. The nodes of this tree represent the GUIObjects contained in the actually constructed webpage. This class also provides a Tree-CellRenderer to implement a special look&feel for this tree.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignGUITree(WebpageModel webpageModel) Creates an instance of the DesignGUITree.</td>
</tr>
</tbody>
</table>
\end{verbatim}
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean allParentsPlaced(DefaultMutableTreeNode node)</td>
<td>This method checks whether or not all of the parents of the given node are already placed on the page or not.</td>
</tr>
<tr>
<td>public Enumeration getChildren(GUIObject object)</td>
<td>This method returns the children-nodes of the node belonging to the given GUIObject node first.</td>
</tr>
<tr>
<td>public DefaultMutableTreeNode getRootNode()</td>
<td>Returns the root node.</td>
</tr>
<tr>
<td>public DefaultMutableTreeNode getSelectedNode()</td>
<td>Returns the actually selected node.</td>
</tr>
<tr>
<td>public void insertNode(MutableTreeNode parent, MutableTreeNode child)</td>
<td>Inserts a given child-node into the existing tree as a child of the given parent-node.</td>
</tr>
<tr>
<td>public boolean isPlacedOnPage()</td>
<td>This simple method checks whether or not the selected node is already placed on the page or not. This determines the color of the node and the behavior of the editor!</td>
</tr>
<tr>
<td>public boolean isPlacedOnPage(GUIObject obj)</td>
<td>This simple method checks whether or not the given GUIObject is already placed on the page or not. This determines the color of the node and the behavior of the editor.</td>
</tr>
<tr>
<td>public void reloadNode(DefaultMutableTreeNode node)</td>
<td>Reloads the given node. This means, the method load is called again to update the tree if any changes have occurred.</td>
</tr>
<tr>
<td>public void removeNode(MutableTreeNode node)</td>
<td>The given node is removed from the tree.</td>
</tr>
<tr>
<td>public void selectGUIObject(GUIObject object)</td>
<td>This method selects the DefaultMutableTreeNode which represents the given GUIObject. In order to do this we must find the corresponding node first.</td>
</tr>
<tr>
<td>public void selectNode(MutableTreeNode node)</td>
<td>Selects the given node from the tree.</td>
</tr>
</tbody>
</table>

C.2.2 layout.DesignGUITree DesignTreeCellRenderer

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
javax.swing.tree.DefaultTreeCellRenderer

public DesignGUITree.DesignTreeCellRenderer
extends DefaultTreeCellRenderer
C.2. LAYOUT

Renders the entries of the Design Tree depending on their reference counting. Containers that are used more than once are displayed in italics.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignGUITree.DesignTreeCellRenderer() Creates an instance of the cellrenderer for this DesignGUITree.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Component</td>
<td>getTreeCellRendererComponent(JTree tree, Object value, boolean sel, boolean expanded, boolean leaf, int row, boolean hasFocus) Configures the renderer based on the passed in components. The value is set from messaging the tree with convertValueToText, which ultimately invokes toString on value. The foreground color is set based on the selection and the icon is set based on leaf and expanded.</td>
</tr>
</tbody>
</table>

C.2.3 layout.DesignGUITreeSelectionListener

java.lang.Object

public DesignGUITreeSelectionListener
extends Object
implements TreeSelectionListener

This listener manages every action the user takes in the structure-tree of a single page. This tree is only shown while we are in the editing page mode.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignGUITreeSelectionListener(DesignGUITree designTree, WebpageModel webpageModel) Creates a new instance of SimpleGUIlayoutTreeSelectionListener.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public GUIObject</td>
<td>getSelectedObject() Returns the GUIObject represented by the selected TreeNode.</td>
</tr>
</tbody>
</table>
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>valueChanged(TreeSelectionEvent e) Invoked when another in the tree was selected.</td>
</tr>
</tbody>
</table>

C.2.4 layout.DesignTreePopupActionHandler

```java
public DesignTreePopupActionHandler
extends Object
implements ActionListener, PopupActionHandler
```

This class handles actions in the popup-menu od the designTree. It shows the allowed actions and calls the if they are selected.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignTreePopupActionHandler(WebpageModel webpageModel, DesignGUITree designTree, XSLGUIPanel mainPanel, ResourceManager resourceManager) Creates an instance of DesignTreePopupActionHandler</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>actionPerformed(ActionEvent e) Invokes the Action of a certain menu. First we check wether the action is possible or not.</td>
</tr>
<tr>
<td>public void</td>
<td>showPopupMenu(Point p) This method shows a popup menu for the selected node and shows it on the given coordinate.</td>
</tr>
</tbody>
</table>

C.2.5 layout.PopupActionHandler

```java
public interface PopupActionHandler
```

This interface defines methods which classes, who want to show and handle a popupmenu on a given point, must implement.

Methods
C.2. LAYOUT

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>showPopupMenu(Point p) This method is called to show a popupmenu on the given point.</td>
</tr>
</tbody>
</table>

C.2.6 layout.SimpleGUILayoutTree

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JTree
structure.GUIStructureTree

public SimpleGUILayoutTree
extends GUIStructureTree

Provides the tree used in the "GUI Layout"-Tab. The SimpleGUILayout tree is an extension of the structure tree. It shows all objects in the hierarchy, beginning at the directory that is ‘root’. Objects like pages, collections, subdirectories, etc. are shown. Furthermore all objects that are contained in an attribute are also shown in the tree.

To provide the user with a fast feedback about the templates already defined, the entries in the tree get three different colors, depending on their state. Green means: “everything ok”, orange means: “template defined”, but one of the children is not green yet, and red means: “No template defined” For most objects two different templates can be defined, the “DefaultTemplate” and the “ContextTemplate”. If none of them is defined, the entry is colored red, and all parent-entries will be colored orange or red. Except of the used icons for the roots and the event-handling, the SimpleGUILayoutTree is equivalent to the GUILayoutTree developed by S. Zweifel.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleGUILayoutTree(GUIDirectory rootdir) Creates an instance of SimpleGUILayoutTree</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>reloadNode(DefaultMutableTreeNode node) Reloads the given node. Normally used to react on changes in the structure.</td>
</tr>
</tbody>
</table>
C.2.7 layout.SimpleGUIsetLayoutTree.LayoutTreeCellRenderer

```java
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
javax.swing.tree.DefaultTreeCellRenderer
```

```java
public SimpleGUIsetLayoutTree.LayoutTreeCellRenderer
extends DefaultTreeCellRenderer
```

Renders the entries of the Design Tree depending on their reference counting. Containers that are used more than once are displayed in italics.

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleGUIsetLayoutTree.LayoutTreeCellRenderer()</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Component</td>
<td>getTreeCellRendererComponent(JTree tree, Object value, boolean sel, boolean expanded, boolean leaf, int row, boolean hasFocus)</td>
</tr>
</tbody>
</table>

C.2.8 layout.SimpleGUIsetLayoutTreeSelectionListener

```java
java.lang.Object
```

```java
public SimpleGUIsetLayoutTreeSelectionListener
extends Object
implements TreeSelectionListener
```

This class is a TreeSelectionListener for a SimpleGUIsetLayoutTree. Its very simple purpose is to activate and deactivate the correct buttons (Load layout and Create new page layout) depending on the nodes selected.

**Constructors**
C.2. LAYOUT

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleGUILayoutTreeSelectionListener(SimpleGUIlayoutTree layoutTree, XSLGUIPanel mainPanel) Creates a new instance of SimpleGUIlayoutTreeSelectionListener</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>valueChanged(TreeSelectionEvent e) Invoked when through a change of the selected node a TreeSelection-Event is fired. It handles the activation and deactivation of the buttons in the mainpanel according to the selected node.</td>
</tr>
</tbody>
</table>

C.2.9 layout.TemplateComponentAssociationEraser

java.lang.Object

public TemplateComponentAssociationEraser extends Object

This small class provides a way to delete associations between CMComponent and CMTemplates. Its needed to make sure that only one template is associated to a component at the moment.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemplateComponentAssociationEraser(GUIContainer webpage, ResourceManager resourceManager) Creates a new instance of TemplateComponentAssociationEraser</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>startErasingAssociations() Starts the erasing the associations!</td>
</tr>
<tr>
<td>public void</td>
<td>startErasingTemplates() Starts erasing the templates</td>
</tr>
</tbody>
</table>

C.2.10 layoutTemplateNameDialog

java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Dialog
javax.swing.JDialog

class TemplateNameDialog extends JDialog

A very simple dialog to read in the name, the user wants to give to the template he creates!

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemplateNameDialog(Frame parent, ResourceManager resourceManager, int webpageWidth, boolean modal) Creates new form TemplateNameDialog</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public String</td>
<td>getTemplateName() Returns the name entered in the templateName textfield</td>
</tr>
<tr>
<td>public int</td>
<td>getWebpageWidth() Returns true when the webpage should have a fixed width</td>
</tr>
<tr>
<td>public boolean</td>
<td>isCanceled() Returns true when the ok-button was not pressed</td>
</tr>
<tr>
<td>public boolean</td>
<td>isFixedWidth() Returns true when the webpage should have a fixed width</td>
</tr>
</tbody>
</table>

C.2.11 layout.XSLGUIPanel

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel

class XSLGUIPanel extends JPanel

This class is the main panel for graphical template editor. It consist of a menubar to the top of the panel. In the menu are a few buttons to load, save, cancel or create a new layout. All these actions are invoked in this class and then passed to other classes for handling. To the right of these buttons is some space left for the dynamic property-toolbars associated with each WebpageObject. These toolbars are suitable to change properties like fonts and colors.

On the left side of this panel is a tree, depending on the current action, the tree shows the structure of the whole website or only the structure of the actually edited page.
The largest component in this panel is the webpage-panel. It represents the currently edited webpage. It lies below the property-toolbar and to the right of the tree. All the layouting is done within this panel.

### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLGUIPanel() Creates new form XSLGuiPanel</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>activatejButtonLoad() Activates the new page button. This should only be done when a page-object is selected</td>
</tr>
<tr>
<td>public void</td>
<td>activatejButtonNewPageLayout() Activates the new page button. This should only be done when a page-object is selected</td>
</tr>
<tr>
<td>public void</td>
<td>deactivatejButtonLoad() deactivates the new page button. This should be done when no page-object is selected or when we already edit a page.</td>
</tr>
<tr>
<td>public void</td>
<td>deactivatejButtonNewPageLayout() deactivates the new page button. This should be done when no page-object is selected or when we already edit a page.</td>
</tr>
<tr>
<td>public DesignGUITree</td>
<td>getDesignGUITree() Returns the actual DesignGUI-Tree.</td>
</tr>
<tr>
<td>public DesignTreePopupActionHandler</td>
<td>getPopupActionHandler() Returns the popupListener associated with the actual DesignGUITree.</td>
</tr>
<tr>
<td>public Webpage</td>
<td>getWebpage() Returns the page currently edited. Null if no page is being edited</td>
</tr>
<tr>
<td>public DesignGUITree</td>
<td>initPageStructureTree(WebpageModel webpage-Model) This method creates a tree with the structure of the webpage represented by the given webpage-Model</td>
</tr>
<tr>
<td>public boolean</td>
<td>isEditingPage() Returns wether or not a page is edited at the moment.</td>
</tr>
<tr>
<td>public void</td>
<td>removePropertyToolBar() This method just simply removes any component in the JPanel jPanelToolbar! Its called before a new PropertyToolBar is set. The programmer should call it as well when there is no selected object anymore!</td>
</tr>
<tr>
<td>public void</td>
<td>removePropertyToolBar(JPanel propertyToolBar) This method removes the given JPanel from the JPanel jPanelToolBar! The programmer should call this method when a WebpageObject is deselected!</td>
</tr>
<tr>
<td>public void</td>
<td>setPropertyToolBar(JPanel propertyToolBar) This method places the given PropertyToolBar in the top-menupanel</td>
</tr>
</tbody>
</table>
C.3 layout.webpage

C.3.1 layout.webpage.Webpage

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLayeredPane
layout.webpage.WebpageContainer

public Webpage
extends WebpageContainer

Represents the webpage on the screen. Here, the actual drawing and designing is done. In the data model, the webpage is a CMContainer, therefore this class subclasses WebpageContainer.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Webpage(CMComponent cmComponent, String pageName, String actualTemplateName, BuildParameters parameters, int webpageWidth, boolean fixedWidth) Creates a new instance of type Webpage.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>drawContent() Draws this component on the page. This Method assumes that the size and the position of the component are already defined. The component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td>drawContent(Dimension size, Point location) Draws this component on the page. This Method is responsible to set position and size of the component and to get its content! For the Webpage object, this method is empty because all the drawing is done in the WebpageObjects contained within the page.</td>
</tr>
<tr>
<td>public String</td>
<td>getActualTemplateName() Returns the Name of the template generated at the moment!</td>
</tr>
<tr>
<td>public CMComponent</td>
<td>getCMComponent() Returns the CMContainer that is represented by this Webpage object</td>
</tr>
<tr>
<td>public String</td>
<td>getLayoutTemplateFileName() Returns the XSL-template needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public int</td>
<td>getWebpageWidth() Returns the width of the webpage.</td>
</tr>
<tr>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>public boolean</td>
<td><strong>isFixedWidth()</strong> Returns wether or not a fixed width is used.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject</strong> (Element webpageObjectXMLTemplate) Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setDropTarget</strong> (DesignGUITree designTree, ResourceManager resourceManager) Sets the DragTarget needed to invoke the drag&amp;drop operations on the webpage. Also sets the property-Toolbar.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setFixedWidth</strong> (boolean fixedWidth) Sets wether or not a fixed width should be used.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setWebpageWidth</strong> (int webpageWidth) Sets the width this webpage should have if the fixedWidth property is set.</td>
</tr>
</tbody>
</table>

### C.3.2 layout.webpage.WebpageCollectionContainer

```java
public WebpageCollectionContainer extends WebpageContainer
```

This class represents the CMCollection type. A CMCollection is just simple a CMContainer without any specialities. Most of the Method in this class are already defined in its superclass WebpageContainer. The only important method implemented here is drawContent!

#### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WebpageCollectionContainer</strong> (CMComponent cmComponent, WebpageContainer container, WebpageModel webpageModel, BuildParameters parameters, String name, int webpageWidth, ResourceManager resourceManager) Creates a new instance of WebpageCollectionContainer</td>
</tr>
</tbody>
</table>

#### Methods
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td><code>drawContent()</code> Draws the container. This abstract method must be implemented by all subclasses. The size and coordinates for this method must be set already. The idea is that the concrete subclasses know absolutely well how it should be drawn!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent(Dimension size, Point location)</code> Draws the container. This abstract method must be implemented by all subclasses. Each subclass knows absolutely well how it should be drawn!</td>
</tr>
<tr>
<td>public String</td>
<td><code>getLayoutTemplateFileName()</code> Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
</tbody>
</table>

C.3.3 `layout.webpage.WebpageComponent`

    java.lang.Object
    java.awt.Component
    java.awt.Container
    javax.swing.JComponent
    javax.swing.JLabel

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
</table>

This abstract class implements the WebpageObject interface, it is the superclass of every instance representing a CMComponent which is not a CMContainer. These are types like CMLink, CMText, CMPicture, CMSitemap and CMNavigation. They contain usually text or images, therefore this class extends JLabel which suits perfectly for this content. For every concrete subtype of CMComponent there exists a subclass of this class. An Instance of one of these subclasses should be generated with the WebpageObjectFactory Class!

| Description |

<p>| Methods |</p>
<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><code>calculateDefaultSize(Point location)</code> Given its location this Method calculates the default size of this WebpageObject.</td>
</tr>
<tr>
<td>public abstract Dimension</td>
<td><code>calculateSize(int width)</code> Calculates the size of this WebpageComponent given its width. This method is useful with text, as the height of a text depends on its width!</td>
</tr>
<tr>
<td>public void</td>
<td><code>deselectContent()</code> Deselects the component. This means its colored bound vanishes</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getBackgroundColor()</code> Returns the background color of this WebpageObject.</td>
</tr>
<tr>
<td>public BuildParameters</td>
<td><code>getBuildParameters()</code> The Buildparameters for this WebpageObject</td>
</tr>
<tr>
<td>public CMComponent</td>
<td><code>getCMComponent()</code> The CMComponent represented by this WebpageObject.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getMoveColor()</code> The color for the border while moving.</td>
</tr>
<tr>
<td>public Border</td>
<td><code>getNormalBorder()</code> The normally used border of this WebpageObject.</td>
</tr>
<tr>
<td>public WebpageContainer</td>
<td><code>getParentWebpageContainer()</code> The WebpageContainer containing this WebpageObject.</td>
</tr>
<tr>
<td>public JPanel</td>
<td><code>getPropertyToolbar()</code> gets the property-toolbar for this specific WebpageObject</td>
</tr>
<tr>
<td>public Border</td>
<td><code>getSelectBorder()</code> Returns the border used when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getUnselectColor()</code> Returns the normal color of the selection-shade.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getUnselectColor()</code> Returns the normal color of the border.</td>
</tr>
<tr>
<td>public String</td>
<td><code>getWebpageObjectName()</code> The name of this WebpageObject. Normally this is the name of the CMComponent represented.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>intersects(Rectangle bound)</code> Checks wether or not the given bound of this content would intersects with other contents within the same parentcontainer.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isForbiddenPosition()</code> Returns wether or not the component lies on a forbidden position. Forbidden means, it has an intersection or it does not fit in there!</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isMoving()</code> Returns wether or not the WebpageObject is moving at the moment.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isPlaced()</code> Returns if the WebpageObject is already placed on the page or not.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isSelected()</code> Returns wether or not this WebpageObject is selected by the user.</td>
</tr>
<tr>
<td>public void</td>
<td><code>loadWebpageObject(Element webpageObjectXML-Template)</code> Loads the properties of this WebpageObject from the given Element</td>
</tr>
</tbody>
</table>
### APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void paintComponent(Graphics g)</code></td>
<td>Method called by swing to paint this WebpageComponent. Overrides the method of JLabel to be able to show a shadow when this WebpageComponent is selected.</td>
</tr>
<tr>
<td><code>public void saveToDatabase(Element template, String templateName)</code></td>
<td>Writes the given template to the database and associates it with this WebpageObject. This method is called for all WebpageObjects except on of the type Webpage. For Webpage objects, there is a special method <code>saveWebpageToDatabase()</code></td>
</tr>
<tr>
<td><code>public void selectContent()</code></td>
<td>Selects the component. This means its bound is colored.</td>
</tr>
<tr>
<td><code>public void setBackgroundColor(Color backgroundColor)</code></td>
<td>Sets the background color of this WebpageObject.</td>
</tr>
<tr>
<td><code>public void setBuildParameters(BuildParameters parameters)</code></td>
<td>Sets the Buildparameters for this WebpageObject</td>
</tr>
<tr>
<td><code>public void setForbiddenPosition(boolean forbiddenPosition)</code></td>
<td>Sets wether or not the WebpageObject lies on a forbidden position.</td>
</tr>
<tr>
<td><code>public void setLocation(int x, int y)</code></td>
<td>Sets the position of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td><code>public void setLocation(Point p)</code></td>
<td>Sets the position of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td><code>public void setMoveColor(Color moveColor)</code></td>
<td>The color of the border when the WebpageObject is moving!</td>
</tr>
<tr>
<td><code>public void setMoving(boolean moving)</code></td>
<td>Sets wether or not the WebpageObject is moving.</td>
</tr>
<tr>
<td><code>public void setPlaced(boolean placed)</code></td>
<td>Sets wether or not the WebpageObject is placed on the Webpage already.</td>
</tr>
<tr>
<td><code>public void setPropertyToolbar(JPanel propertyToolbar)</code></td>
<td>Sets the property-toolbar for this specific WebpageObject.</td>
</tr>
<tr>
<td><code>public void setSelectColor(Color selectColor)</code></td>
<td>Sets the Color of the shade shown when this color is selected!</td>
</tr>
<tr>
<td><code>public void setSelected(boolean selected)</code></td>
<td>Sets wether or not this WebpageObject is selected.</td>
</tr>
<tr>
<td><code>public void setSelectStroke(BasicStroke selectStroke)</code></td>
<td>Sets the stroke to draw the border when the WebpageObject is selected.</td>
</tr>
<tr>
<td><code>public void setSize(Dimension d)</code></td>
<td>Sets the size of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td><code>public void setSize(int width, int height)</code></td>
<td>Sets the size of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
</tbody>
</table>
C.3. LAYOUT.WEBPAGE

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>setUnselectColor(Color unselectColor) Sets the color for the border if the WebpageObject is not selected.</td>
</tr>
<tr>
<td>public String</td>
<td>transformXSLStylesheet(String xml) This method creates the XSL-template out of the given XML file. It returns the stylesheet as a String which makes it easy to insert it into the database!</td>
</tr>
</tbody>
</table>

C.3.4 layout.webpage.WebpageComponentMouseHandler

```
java.lang.Object
java.awt.event.MouseAdapter
```

public WebpageComponentMouseHandler
extends MouseAdapter

WebpageComponentMouseHandler is used on the panel that represents the webpage to handle mouse events corresponding to clicking on a WebpageComponent and to invoke the needed action and/or response

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageComponentMouseHandler(WebpageComponent webpageComponent) Creates a new instance of WebpageContainerMouseHandler</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>mouseClicked(MouseEvent e) Invoked when the mouse is clicked. 2 Actions are possible: 1. Show the popup menu 2. A selection event, select the webpage-Component!</td>
</tr>
<tr>
<td>public void</td>
<td>mouseEntered(MouseEvent e) Invoked when the mouse enters the WebpageComponent</td>
</tr>
<tr>
<td>public void</td>
<td>mouseExited(MouseEvent e) Invoked when the mouse exits the WebpageComponent</td>
</tr>
<tr>
<td>public void</td>
<td>mousePressed(MouseEvent e) Invoked when the mouse is pressed. This simple method just checks weather the event should show a popup menu or not.</td>
</tr>
<tr>
<td>public void</td>
<td>mouseReleased(MouseEvent e) Invoked when the mouse is released. Two actions are possible: 1. Show the popup-menu 2. If the webpageComponent was moved the operations to finish the moving must be invoked.</td>
</tr>
</tbody>
</table>
C.3.5 layout.webpage.WebpageComponentMouseMotionHandler

java.lang.Object
java.awt.event.MouseMotionAdapter

public WebpageComponentMouseMotionHandler
extends MouseMotionAdapter

This class handles everything concerning the moving and resizing of a WebpageComponent. Every instance of WebpageComponent has its own WebpageComponentMouseMotionHandler waiting for events.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageComponentMouseMotionHandler(WebpageComponent webpageComponent) Creates a new instance of WebpageComponentMouseMotionHandler</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int</td>
<td>getResizeDirection() Returns the Code for one of the eight direction a resizing operation can take place.</td>
</tr>
<tr>
<td>public Point</td>
<td>getStartLocation() Gets the start-location of a resizing or dragging operation.</td>
</tr>
<tr>
<td>public Dimension</td>
<td>getStartSize() Gets the size of the component when the operation was started.</td>
</tr>
<tr>
<td>public boolean</td>
<td>isDragged() Returns wether or not the webpageComponent is dragged at the moment.</td>
</tr>
<tr>
<td>public boolean</td>
<td>isForbiddenPosition() Returns wether or not the webpageComponent has a forbiddenPosition at the moment</td>
</tr>
<tr>
<td>public boolean</td>
<td>isRelocationSuccessful() Returns wether or not the relocation was successful.</td>
</tr>
<tr>
<td>public void</td>
<td>mouseDragged(MouseEvent e) Invoked when the mouse is dragged. The method manages the dragging of the webpageComponent around its container. Depending on the concrete direction of the movement it invokes a specialized method to do the actual dragging.</td>
</tr>
<tr>
<td>public void</td>
<td>mouseMoved(MouseEvent e) Invoked when the mouse is moved. Checks wether the mouse entered or exited the webpageComponent.</td>
</tr>
<tr>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>public void</td>
<td>resetResizeDirection() This method is called to reset the cursor when the mouse quits the sensitive part of the webpageComponent.</td>
</tr>
<tr>
<td>public void</td>
<td>setDragged(boolean dragged) Sets wether or not a drag-operation is going on.</td>
</tr>
<tr>
<td>public void</td>
<td>setEntered(boolean entered) Sets wether or not the cursor has entered the webpageComponent</td>
</tr>
<tr>
<td>public void</td>
<td>setExited(boolean exited) Sets wether or not the cursor has exited the webpageComponent</td>
</tr>
<tr>
<td>public void</td>
<td>setForbiddenPosition(boolean forbiddenPosition) Sets wether or not the webpageComponent has a forbiddenPosition at the moment</td>
</tr>
<tr>
<td>public void</td>
<td>setRelocationSuccessful(boolean relocationSuccessful) Sets wether or not the relocation was a success.</td>
</tr>
</tbody>
</table>

### C.3.6 layout.webpage.WebpageContainer

```java
public abstract WebpageContainer
extends JLayeredPane
implements WebpageObject
```

This abstract class implements the WebpageObject interface, it is the superclass of every instance representing a CMContainer. These are types like CMMixed, CMCollection. They contain other CMComponents, therefore this class extends JPanel which suits perfectly for this content. For every concrete subtype of CMContainer exists a subclass of this class. An Instance of one of these subclasses should be generated with the WebpageObjectFactory Class!

#### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageContainer(CMComponent cmComponent, WebpageContainer parentContainer, WebpageModel webpageModel, boolean page, String webpageContainerName, int webPageWidth) Creates a new instance of WebpageContainer The page variable states wether or not this container is a page!</td>
</tr>
</tbody>
</table>

#### Methods
# Appendix C. The API of the Graphical Template Editor

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void addWebpageComponent(WebpageComponent component)</td>
<td>Adds the given WebpageComponent to this WebpageContainer.</td>
</tr>
<tr>
<td>public void addWebpageContainer(WebpageContainer container)</td>
<td>Adds the given WebpageContainer to this WebpageContainer.</td>
</tr>
<tr>
<td>public void addWebpageObject(WebpageComponent content, Point location, Dimension size)</td>
<td>adds a WebpageComponent-Object to a page.</td>
</tr>
<tr>
<td>public Dimension calculateDefaultSize(Point location)</td>
<td>Given its location, this Method calculates the default size of this Content.</td>
</tr>
<tr>
<td>public Element createXMLLayout()</td>
<td>This method calls the layout-generator which creates the XML-Element that holds informations needed to generate the XSL-template for the layout of this container.</td>
</tr>
<tr>
<td>public void deselectContent()</td>
<td>deselects the component. This means its colored shape vanishes.</td>
</tr>
<tr>
<td>public void drawAllChildren(Point location)</td>
<td>This method is called to add all children to the container and to set their size and position.</td>
</tr>
<tr>
<td>public Color getBackgroundColor()</td>
<td>Returns the background color of this WebpageObject.</td>
</tr>
<tr>
<td>public BuildParameters getBuildParameters()</td>
<td>The Buildparameters for this WebpageObject</td>
</tr>
<tr>
<td>public Color getChildSelectedColor()</td>
<td>This method gets the color this WebpageContainer should be shaded with if a child WebpageObject is selected. At the moment this function is not used.</td>
</tr>
<tr>
<td>public CMComponent getCMComponent()</td>
<td>The CMComponent represented by this WebpageObject.</td>
</tr>
<tr>
<td>public int getFixedWebpageContainerWidth()</td>
<td>Returns the width of this container relative to the total width of the webpage. This method is useful when the user wants to create a webpage with a fixed width!</td>
</tr>
<tr>
<td>public int getFixedWebpageWidth()</td>
<td>Returns the width of the Webpage if the fixedWidth-property of the webpage is true.</td>
</tr>
<tr>
<td>public Color getMoveColor()</td>
<td>The color for the border while moving.</td>
</tr>
<tr>
<td>public Border getNormalBorder()</td>
<td>The normally used border of this WebpageObject.</td>
</tr>
<tr>
<td>public WebpageContainer getParentWebpageContainer()</td>
<td>The WebpageContainer containing this WebpageObject.</td>
</tr>
<tr>
<td>public JPanel getPropertyToolbar()</td>
<td>gets the property-toolbar for this specific WebpageObject</td>
</tr>
<tr>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>public Dimension</td>
<td>getProvisoricSize() Returns the provisoric calculated size of this container. This size is used to do calculations with this WebpageContainer before it is actually placed on the screen.</td>
</tr>
<tr>
<td>public Border</td>
<td>getSelectBorder() Returns the border used when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public Color</td>
<td>getSelectColor() Returns the color of the selection-shade.</td>
</tr>
<tr>
<td>public BasicStroke</td>
<td>getSelectStroke() Gets the stroke to draw the border when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public int</td>
<td>getSpacer() Returns the spacer-value of this container. The spacer value is the default amount of blank space between the border of this container and the border of its children.</td>
</tr>
<tr>
<td>public int</td>
<td>getSpacerFactor() Returns the number of spacer-values of this container. The spacer value is the default amount of blank space between the border of this container and the border of its children.</td>
</tr>
<tr>
<td>public Color</td>
<td>getUnselectColor() Returns the normal color of the border.</td>
</tr>
<tr>
<td>public Vector</td>
<td>getWebpageComponents() Returns the Webpage-Components contained within this WebpageContainer.</td>
</tr>
<tr>
<td>public WebpageContainer</td>
<td>getWebpageContainerByName(String name) Look is in the collection of contained WebpageContainers for the Object with the given Name.</td>
</tr>
<tr>
<td>public Vector</td>
<td>getWebpageContainers() Returns the WebpageContainers contained within this WebpageContainer.</td>
</tr>
<tr>
<td>public WebpageModel</td>
<td>getWebpageModel() Returns the model of the Webpage-Object which contains this WebpageContainer.</td>
</tr>
<tr>
<td>public String</td>
<td>getWebpageObjectName() Returns the name of this WebpageContainer.</td>
</tr>
<tr>
<td>public boolean</td>
<td>intersects(Rectangle bound) Checks wether or not the given “new” bound of this content would intersects with other contents within the same parentcontainer.</td>
</tr>
<tr>
<td>public boolean</td>
<td>isChildSelected() Returns wether or not a child of this WebpageContainer is selected</td>
</tr>
<tr>
<td>public boolean</td>
<td>isForbiddenPosition() Returns wether or not the component lies on a forbidden position. Forbidden means, it has an intersection or it does not fit in there!</td>
</tr>
<tr>
<td>public boolean</td>
<td>isMoving() Returns wether or not the WebpageObject is moving at the moment.</td>
</tr>
<tr>
<td>public boolean</td>
<td>isPage() Returns wether or not this WebpageContainer is also a Webpage.</td>
</tr>
<tr>
<td>public boolean</td>
<td>isPlaced() Returns if the WebpageObject is already placed on the page or not.</td>
</tr>
<tr>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| public boolean
isSelected() | Returns whether or not this WebpageObject is selected by the user.                                                                           |
| public boolean
isShowGrid()   | Returns whether or not the aligning grid is visible.                                                                                       |
| public void
loadWebpageObject(Element webpageObjectXMLTemplate) | Loads the properties of this WebpageObject from the given Element.                                                                           |
| public void
paintComponent(Graphics g) | Overrides the swing-method to paint this JPanel. Overridden to show the selection-shade and to show the grid if it is desired. |
| public boolean
relocateContent(Point newLocation, Dimension newSize) | Relocates and resizes the WebpageObject to the given location and size. This method can fail when the WebpageObject cannot be placed on its new location because of the size or because of an intersection! Returns true if it is possible to resize the content. For the WebpageContainer class The result of this method depends on whether or not the children of the container fit into its new size! |
| public void
removeWebpageContainer() | "Removes" the WebpageContainer from its parent. Actually, it is not really removed. It is only hidden through setVisible(false); |
| public void
restoreChildPosition() | This Method restores the actual position of each child of the container. This is needed when the container should be resized but the resizing operation failed and the old must be restored. |
| public void
restoreChildPosition(Point offsetLocation) | This Method restores the actual position of each child of the container. It adds the given offsetLocation to the originally stored values. |
| public void
saveToDatabase(Element template, String templateName) | Writes the given template to the database and associates it with this WebpageObject. This method is called for all WebpageObjects except of the type Webpage. For Webpage objects, there is a special method saveWebpageToDatabase() |
| public void
selectContent() | Selects the container. This means its shape is colored |
| public void
setBackgroundColor(Color backgroundColor) | Sets the background color of this container. Also ask if the background color of the children should be changed as well! |
| public void
setBackgroundColor(Color backgroundColor, boolean propagateChange) | Sets the background color of this container. If propagateChange is true, the background color of children is changed as well! |
| public void
setBuildParameters(BuildParameters parameters) | Sets the Buildparameters for this WebpageObject |
<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setChildSelected(boolean childSelected)</td>
<td>Sets whether or not a child of this WebpageContainer is selected.</td>
</tr>
<tr>
<td>public void setChildSelectedColor(Color childSelectColor)</td>
<td>This method sets the color this WebpageContainer should be shaded with if a child WebpageObject is selected. At the moment this function is not used, but the color can be set!</td>
</tr>
<tr>
<td>public void setForbiddenPosition(boolean forbiddenPosition)</td>
<td>Sets whether or not the WebpageObject lies on a forbidden position.</td>
</tr>
<tr>
<td>public void setLocation(int x, int y)</td>
<td>Sets the position of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td>public void setLocation(Point p)</td>
<td>Sets the position of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td>public void setMoveColor(Color moveColor)</td>
<td>The color of the border when the WebpageObject is moving!</td>
</tr>
<tr>
<td>public void setMoving(boolean moving)</td>
<td>Sets whether or not the WebpageObject is moving.</td>
</tr>
<tr>
<td>public void setPlaced(boolean placed)</td>
<td>Sets whether or not the WebpageObject is placed on the Webpage already.</td>
</tr>
<tr>
<td>public void setPropertyToolbar(JPanel propertyToolbar)</td>
<td>Sets the property-toolbar for this specific WebpageObject.</td>
</tr>
<tr>
<td>public void setSelectColor(Color selectColor)</td>
<td>Sets the Color of the shade shown when this color is selected!</td>
</tr>
<tr>
<td>public void setSelected(boolean selected)</td>
<td>Sets whether or not this WebpageObject is selected.</td>
</tr>
<tr>
<td>public void setSelectedContent(WebpageObject selectedContent)</td>
<td>Sets the selected WebpageObject if it is within this container.</td>
</tr>
<tr>
<td>public void setSelectStroke(BasicStroke selectStroke)</td>
<td>Sets the stroke to draw the border when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public void setShowGrid(boolean showGrid)</td>
<td>Sets whether or not the aligning grid is visible.</td>
</tr>
<tr>
<td>public void setSize(Dimension d)</td>
<td>Sets the size of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td>public void setSize(int width, int height)</td>
<td>Sets the size of this WebpageComponent. Overrides the method of JLabel in order to make sure the size is aligned to 10.</td>
</tr>
<tr>
<td>public void setUnselectColor(Color unselectColor)</td>
<td>Sets the color for the border if the WebpageObject is not selected.</td>
</tr>
</tbody>
</table>
### Returns

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setWebpageModel(WebpageModel webpageModel)</td>
<td>Sets the webpageModel which manages the page this container belongs to. It's necessary to actualize the webpageModel of the corresponding Mousehandler</td>
</tr>
<tr>
<td>public int spaceToTheRight(Point p)</td>
<td>Calculates how much spaces to the right side of the given point is still left for this container.</td>
</tr>
<tr>
<td>public void storeChildPosition()</td>
<td>This method stores the actual position of each child of the container. The method is called before the container should be resized in order to be able to restore the old state.</td>
</tr>
<tr>
<td>public String transformXSLStylesheet(String xml)</td>
<td>This method creates the XSL-template out of the given XML file. It returns the stylesheet as a String which makes it easy to insert it into the database!</td>
</tr>
<tr>
<td>public void unselectChildContent()</td>
<td>If the deselected WebpageObject is a child of this container, this method should be called to set the selected WebpageObject to null.</td>
</tr>
</tbody>
</table>

### C.3.7 `layout.webpage.WebpageContainerMouseHandler`

```java
java.lang.Object
java.awt.event.MouseAdapter
```

public WebpageContainerMouseHandler extends MouseAdapter

WebpageContainerMouseHandler is used on the panel that represents the webpage to handle mouse events corresponding to clicking on a WebpageContainer and to invoke the needed action and/or response.

#### Constructors

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageContainerMouseHandler(WebpageContainer webpageContainer)</td>
<td>Creates a new instance of WebpageContainerMouseHandler</td>
</tr>
</tbody>
</table>

#### Methods
<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void mouseClicked(MouseEvent e)</td>
<td>Invoked when the mouse is clicked. 2 Actions are possible: 1. Show the popup menu 2. A selection event, select the webpage-Container!</td>
</tr>
<tr>
<td>public void mouseEntered(MouseEvent e)</td>
<td>Invoked when the mouse enters the WebpageContainer</td>
</tr>
<tr>
<td>public void mouseExited(MouseEvent e)</td>
<td>Invoked when the mouse exits the WebpageContainer</td>
</tr>
<tr>
<td>public void mousePressed(MouseEvent e)</td>
<td>Invoked when the mouse is pressed. This simple method just checks wether the event should show a popup menu or not.</td>
</tr>
<tr>
<td>public void mouseReleased(MouseEvent e)</td>
<td>Invoked when the mouse is released. Two actions are possible: 1. Show the popup-menu 2. If the webpageContainer was moved, the operations to finish the moving must be invoked.</td>
</tr>
<tr>
<td>public void removeSelf()</td>
<td>Removes itself from the listener-list of its WebpageComponentMouseHandler</td>
</tr>
<tr>
<td>public void setWebpageModel(WebpageModel webpageModel)</td>
<td>Sets the WebpageModel used by the currently edited webpage.</td>
</tr>
</tbody>
</table>

### C.3.8 layout.webpage.WebpageContainerMouseMotionHandler

java.lang.Object
java.awt.event.MouseMotionAdapter

```java
public WebpageContainerMouseMotionHandler extends MouseMotionAdapter
```

This class handles everything concerning the moving and resizing of a WebpageContainer. Every instance of WebpageContainer has its own WebpageContainerMouseMotionHandler waiting for events.

#### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageContainerMouseMotionHandler(WebpageContainer webpageContainer)</td>
</tr>
<tr>
<td>Creates a new instance of WebpageContainerMouseMotionHandler</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int getResizeDirection()</td>
<td>Returns the Code for one of the eight direction a resizing operation can take place.</td>
</tr>
</tbody>
</table>
### Returns

<table>
<thead>
<tr>
<th>Description</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the start-location of a resizing or dragging operation.</td>
<td>getStartLocation()</td>
</tr>
<tr>
<td>Gets the size of the component when the operation was started.</td>
<td>getStartSize()</td>
</tr>
<tr>
<td>Returns whether or not the webpageContainer is dragged at the moment.</td>
<td>isDragged()</td>
</tr>
<tr>
<td>Returns whether or not the webpageContainer has a forbiddenPosition at the moment</td>
<td>isForbiddenPosition()</td>
</tr>
<tr>
<td>Returns whether or not the cursor of the webpageContainer was moved</td>
<td>isMoved()</td>
</tr>
<tr>
<td>Returns whether or not the relocation was successful.</td>
<td>isRelocationSuccessful()</td>
</tr>
<tr>
<td>Invoked when the mouse is dragged. The method manages the dragging of the webpageContainer around its container. Depending on the concrete direction of the movement it invokes a specialized method to do the actual dragging.</td>
<td>mouseDragged(MouseEvent e)</td>
</tr>
<tr>
<td>Invoked when the mouse is moved. Checks whether the mouse entered or exited the webpageContainer.</td>
<td>mouseMoved(MouseEvent e)</td>
</tr>
<tr>
<td>This method is called to reset the cursor when the mouse quits the sensitive part of the webpageContainer.</td>
<td>resetResizeDirection()</td>
</tr>
<tr>
<td>Sets whether or not a drag-operation is going on.</td>
<td>setDragged(boolean dragged)</td>
</tr>
<tr>
<td>Sets whether or not the cursor has entered the webpageContainer.</td>
<td>setEntered(boolean entered)</td>
</tr>
<tr>
<td>Sets whether or not the cursor has exited the webpageContainer.</td>
<td>setExited(boolean exited)</td>
</tr>
<tr>
<td>Sets whether or not the webpageContainer has a forbiddenPosition at the moment.</td>
<td>setForbiddenPosition(boolean forbiddenPosition)</td>
</tr>
<tr>
<td>Sets whether or not the cursor the webpageContainer was moved</td>
<td>setMoved(boolean moved)</td>
</tr>
<tr>
<td>Sets whether or not the relocation was a success.</td>
<td>setRelocationSuccessful(boolean relocationSuccessful)</td>
</tr>
</tbody>
</table>

### C.3.9 layout.webpage.WebpageDefaultComponent

```
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
layout.webpage.WebpageComponent
```
public WebpageDefaultComponent
extends WebpageComponent

This WebpageComponent is used when the actual type of the CMComponent is unknown. This can happen when a user generates a new type. The default component simply shows a rectangle with text indicating its a default component and the name of the unknown type!

 Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WebpageDefaultComponent</strong> (CMComponent cmComponent, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor) Creates a new instance of WebpageTextComponent</td>
</tr>
</tbody>
</table>

 Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><strong>calculateSize</strong> (int width) Returns a default size of the component with the given width. This Method is needed to set the size of the parentContainer correctly.</td>
</tr>
<tr>
<td>public Element</td>
<td><strong>createXMLLayout</strong> () This method creates the XML-Element which stores the layout of the CMComponent represented by this WebpageComponent! With this XML-Element a XSL-template generates then the template to render the Default WebpageComponent on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent</strong> () Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent</strong> (Dimension size, Point location) Draws this component on the page. In the case of the WebpageDefaultComponent the drawing operation is not very complicated. It only shows a default text because nothing about the content of this CMComponent is known.</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getLayoutTemplateFileName</strong> () Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject</td>
</tr>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject</strong> (Element webpageObjectXMLTemplate) Loads the properties of this WebpageObject from the given Element</td>
</tr>
</tbody>
</table>
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean</td>
<td>relocateContent(Point newLocation, Dimension newSize) Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content. If the case of WebpageDefaultComponent I calculate the new minimum size of the content. Then I check for intersections. If there are any, false is returned. Otherwise, the text is reformatted, then content is drawn and finally the method returns true.</td>
</tr>
</tbody>
</table>

C.3.10 layout.webpage.WebpageDefaultContainer

java.lang.Object
does not extend another class

public WebpageDefaultContainer extends WebpageContainer

This WebpageContainer is used when the actual type of the CMContainer is unknown. This can happen when a user generates a new type. The default container simply shows a rectangle with the contained CMComponents!

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageDefaultContainer(CMComponent cmComponent, WebpageContainer container, WebpageModel webpageModel, BuildParameters parameters, String name, int webpageWidth, ResourceManager resourceManager) Creates a new instance of WebpageCollectionContainer</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>drawContent() Draws the container. This abstract method must be implemented by all subclasses. The size and coordinates for this method must be set already. The idea is that the concrete subclasses know absolutely well how it should be drawn!</td>
</tr>
</tbody>
</table>
Returns | Description
---|---
public void | `drawContent(Dimension size, Point location)` Draws the container. This abstract method must be implemented by all subclasses. Each subclass knows absolutely well how it should be drawn! In the case of the default container nothing is done except the drawing of the children.

public String | `getLayoutTemplateFileName()` Returns the name of the XSL-stylesheet needed to generate the template used to create the HTML code for of this WebpageObject.

public void | `loadWebpageObject(Element webpageObjectXMLTemplate)` Loads the properties of this WebpageObject from the given Element.

### C.3.11 `layout.webpage.WebpageMixedContainer`

```
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLayeredPane
layout.webpage.WebpageContainer
```

```java
public WebpageMixedContainer
extends WebpageContainer

The `WebpageMixedContainer` represents the type CMMixed. CMMixed differs from the CMCollection type because it can contain text with markup. This is a problem for the design of the graphical editor as the mixed text is not found in the data model on its own. Therefore when creating a `WebpageMixedContainer` a `WebpageMixedTextComponent` is generated automatically to represent the mixed text.

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| `WebpageMixedContainer(CMMixed cmMixed, WebpageContainer container, WebpageModel webpageModel, BuildParameters parameters, String name, int webpageWidth, ResourceManager resourceManager)` Creates a new instance of WebpageMixedContainer

**Methods**
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td><strong>drawContent()</strong> Draws the container. This abstract method must be implemented by all subclasses. The size and coordinates for this method must be set already. The idea is that the concrete subclasses know absolutely well how it should be drawn!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent(Dimension size, Point location)</strong> Draws the container. This abstract method must be implemented by all subclasses. Each subclass knows absolutely well how it should be drawn! In case of the WebpageMixedTextContainer drawing involves the drawing of its mixed text content. But this is no problem because the WebpageMixedTextComponent can be handled like a normal child.</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getLayoutTemplateFileName()</strong> Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public WebpageMixedTextComponent</td>
<td><strong>getWebpageMixedTextComponent()</strong> Returns the WebpageObject representing the mixed text component of this container</td>
</tr>
</tbody>
</table>

**C.3.12 layout.webpage.WebpageMixedTextComponent**

```java
public WebpageMixedTextComponent extends WebpageComponent
```

This Class represents the mixed text content of the type CMMixed. It is a special subclass of WebpageComponent because there is no type that it represents. The class is very similar to WebpageTextComponent except that it is not possible to change the font-style because this is done by markup in the mixed text.

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageMixedTextComponent()</td>
<td>(CMMixed cmMixed, String mixedContent, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor) Creates a new instance of WebpageTextComponent</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><code>calculateSize(int width)</code> Calculate the size of the component with the given width. This Method is needed to set the size of the parentContainer correctly.</td>
</tr>
<tr>
<td>public Element</td>
<td><code>createXMLLayout()</code> The CMixed-container is a special case. It can contain a text with markup and additional components. This method creates the xml-code for the layout of the textual part of the CMixed Container!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent()</code> Draws this component on the page. This Method assumes that the size and the position of the component are already defined. The component and to get its mixedContent!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent(Dimension size, Point location)</code> Draws this component on the page. In the case of WebpageMixedTextComponent this means preparing and drawing the mixed text.</td>
</tr>
<tr>
<td>public String</td>
<td><code>getContent()</code> Returns the mixed content.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getFontColor()</code> Returns the color of the font.</td>
</tr>
<tr>
<td>public String</td>
<td><code>getLayoutTemplateFileName()</code> Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public Font</td>
<td><code>getTextFont()</code> Returns the font used to draw.</td>
</tr>
<tr>
<td>public void</td>
<td><code>loadWebpageObject(Element webpageObjectXMLTemplate)</code> Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>relocateContent(Point newLocation, Dimension newSize)</code> Relocates and resizes the mixedContent to the given location and size. Returns true if it is possible to resize the mixedContent. In the case of WebpageMixedTextComponent the new minimum size of the mixedContent is calculated. Then comes a check for intersections. If there are any, false is returned. Otherwise, the text is reformatted and the mixedContent is drawn. Finally, if everything succeeded the method returns true.</td>
</tr>
<tr>
<td>public void</td>
<td><code>saveToDatabase(Element template, String templateName)</code> It is necessary to override the method of the superclass because CMixed is a special case of a WebpageComponent because it has no real CMComponent which it represents! Therefore we just give an empty method body.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setFontColor(Color fontColor)</code> Sets the color of the font.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setFontSize(float fontSize)</code> Sets the size of the font</td>
</tr>
<tr>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>public void</td>
<td>setTextFont(Font f) Sets the font used to draw the mixed content</td>
</tr>
</tbody>
</table>

### C.3.13 layout.webpage.WebpageModel

```java
public WebpageModel
extends Object
```

This class represents a webpage as it is seen by the CMServer-project developed by M. Grossniklaus from the ETH Zurich. The WebpageModel manages the collection of all containers and components contained within the webpage. This means it is also the starting point for the creation of these WebpageObjects. Furthermore it is responsible for the selection and deselection of the WebpageObjects a according to the user-input and it manages the cooperation with the DesignGUITree and the other parts of the XCM Sitemanager.

### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageModel(GUIContainer pageGUIContainer, XSLGUIPanel mainPanel, Webpage webpage, ResourceManager resourceManager) Creates an instance of WebpageModel.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public GUIObject</td>
<td>getActualGUIObject() Returns the actually selected GUIObject.</td>
</tr>
<tr>
<td>public WebpageObject</td>
<td>getActualWebpageObject() Returns the actually selected WebpageObject.</td>
</tr>
<tr>
<td>public GUIObject</td>
<td>getGUIObject(WebpageObject webObj) Gets the GUIObject associated with the given WebpageObject.</td>
</tr>
<tr>
<td>public WebpageContainer</td>
<td>getParentWebpageContainer() Returns the WebpageContainer the selected GUIObject belongs to! Returns null if there is no selected GUIObject</td>
</tr>
<tr>
<td>public ResourceManager</td>
<td>getResourceManager() Returns the used ResourceManager</td>
</tr>
<tr>
<td>public Webpage</td>
<td>getWebpage() Returns the Webpage this model represents.</td>
</tr>
<tr>
<td>public GUIContainer</td>
<td>getWebpageGUIContainer() Returns the GUIContainer representing the webpage.</td>
</tr>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject(GUIObject obj) Gets the WebpageObject associated with the given GUIObject.</td>
</tr>
</tbody>
</table>
### C.3. LAYOUT.WEBPAGE

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean</td>
<td>isPlacedOnPage(GUIObject object) This method returns wether or not the given GUIObject is on the page.</td>
</tr>
<tr>
<td>public void</td>
<td>removeActualGUIObject() This method removes the actually selected GUIObject from the page</td>
</tr>
<tr>
<td>public void</td>
<td>selectionChanged(WebpageObject content) This method is called when the user changes the selected WebpageComponent with a mouse-click. It calls the tree to actualize its selection. The DesignGUITreeSelectionListener the calls the setActualGUIObject-Method. It would be nicer with a listener but it suits our purposes.</td>
</tr>
<tr>
<td>public void</td>
<td>setActualGUIObject(GUIObject actualObject, GUIContainer parentObject) Sets the GUIObject which is selected at the moment in the designTree. It also deselects the old actualObject and then selects the new one in the Webpage panel, if its already placed there.</td>
</tr>
<tr>
<td>public void</td>
<td>setDesignGUITree(DesignGUITree designTree) Sets the DesignGUITree. This method is necessary because when the WebpageModel is instantiated, the DesignGUITree is not already created yet. Therefore it must be set after the creation.</td>
</tr>
</tbody>
</table>

### C.3.14 layout.webpage.WebpageNavigationComponent

```java
public WebpageNavigationComponent
extends WebpageComponent
```

This class represents a CMNavigation. As the navigation is created by the CMServer at the moment of the creation of the website we show a dummy placeholder. This has no influence on the functionality of the rendering process which works absolutely fine.

#### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><strong>calculateSize(int width)</strong> Calculates the size of the component with the given width. This Method is needed to set the size of the parentContainer correctly. In this special case we just return a default size.</td>
</tr>
<tr>
<td>public Element</td>
<td><strong>createXMLLayout()</strong> This method creates the XML-Element which stores the layout of the CMNavigation represented by this WebpageComponent! With this XML-Element a XSL-stylesheet generates the template to render the CMNavigation on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent()</strong> Draws this component on the page. This Method assumes that the size and the position of the component are already defined.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent(Dimension size, Point location)</strong> Draws this component on the page.</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getContent()</strong> Gets the content of this CMNavigation</td>
</tr>
<tr>
<td>public Color</td>
<td><strong>setFontColor()</strong> The font color</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getLayoutTemplateFileName()</strong> Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject(Element webpageObjectXMLTemplate)</strong> Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><strong>relocateContent(Point newLocation, Dimension newSize)</strong> Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content. It the case of WebpageNavigationComponent the new minimum size of the content is calculated. A check for intersections follows. If there are any, false is returned. Otherwise, the text is reformatted, then content is drawn and finally the method returns true.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setFontColor(Color fontColor)</strong> Sets the font color</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setTextFont(Font f)</strong> Sets the font of this CMNavigation.</td>
</tr>
</tbody>
</table>

### C.3.15 layout.webpage.WebpageObject

public interface WebpageObject

This interface contains the signatures of all methods that are common to representations of CMComponent in the designer.
### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><code>calculateDefaultSize(Point location)</code> Given its location this Method calculates the default size of this WebpageObject.</td>
</tr>
<tr>
<td>public Element</td>
<td><code>createXMLLayout()</code> creates the XML-representation of the layout of the represented CMComponent.</td>
</tr>
<tr>
<td>public void</td>
<td><code>deselectContent()</code> deselects the component. This means its colored bound vanishes</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent()</code> Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent(Dimension size, Point location)</code> Draws this WebpageObject on the page. This Method is responsible to set position and size of the component and to get its content!</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getBackgroundColor()</code> Returns the background color of this WebpageObject.</td>
</tr>
<tr>
<td>public BuildParameters</td>
<td><code>getBuildParameters()</code> The Buildparameters for this WebpageObject.</td>
</tr>
<tr>
<td>public CMComponent</td>
<td><code>getCMComponent()</code> The CMComponent represented by this WebpageObject.</td>
</tr>
<tr>
<td>public String</td>
<td><code>getLayoutTemplateFileName()</code> Returns the file name of the XSL-template needed to generate the template used to create the html code for of this WebpageObject</td>
</tr>
<tr>
<td>public Point</td>
<td><code>getLocation()</code> Returns the location of this WebpageObject.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getMoveColor()</code> The color for the border while moving.</td>
</tr>
<tr>
<td>public Border</td>
<td><code>getNormalBorder()</code> The normally used border of this WebpageObject.</td>
</tr>
<tr>
<td>public WebpageContainer</td>
<td><code>getParentWebpageContainer()</code> The WebpageContainer containing this WebpageObject.</td>
</tr>
<tr>
<td>public JPanel</td>
<td><code>getPropertyToolbar()</code> gets the property-toolbar for this specific WebpageObject</td>
</tr>
<tr>
<td>public Border</td>
<td><code>getSelectBorder()</code> Returns the border used when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getSelectColor()</code> Returns the color of the selection-shade.</td>
</tr>
<tr>
<td>public Dimension</td>
<td><code>getSize()</code> Returns the size of this WebpageObject.</td>
</tr>
<tr>
<td>public Color</td>
<td><code>getUnselectColor()</code> Returns the normal color of the border.</td>
</tr>
<tr>
<td>public String</td>
<td><code>getWebpageObjectName()</code> The name of this WebpageObject. Normally this is the name of the CMComponent represented.</td>
</tr>
</tbody>
</table>
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean</td>
<td><code>intersects(Rectangle bound)</code> Checks whether or not the given <code>bound</code> of this content would intersects with other contents within the same parentcontainer.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isForbiddenPosition()</code> Returns whether or not the component lies on a forbidden position. Forbidden means, it has an intersection or it does not fit in there!</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isMoving()</code> Returns whether or not the WebpageObject is moving at the moment.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isPlaced()</code> Returns if the WebpageObject is already placed on the page or not.</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isSelected()</code> Returns whether or not this WebpageObject is selected by the user.</td>
</tr>
<tr>
<td>public void</td>
<td><code>loadWebpageObject(Element webpageObjectXMLTemplate)</code> Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>relocateContent(Point newLocation, Dimension newSize)</code> Relocates and resizes the WebpageObject to the given location and size. This method can fail when the WebpageObject cannot be placed on its new location because of the size or because of an intersection! Returns true if it is possible to resize the content.</td>
</tr>
<tr>
<td>public void</td>
<td><code>saveToDatabase(Element webpageObjectXMLTemplate, String templateName)</code> Saves the layout of the given Element to the database!</td>
</tr>
<tr>
<td>public void</td>
<td><code>selectContent()</code> Selects the component. This means its bound is colored</td>
</tr>
<tr>
<td>public void</td>
<td><code>setBackgroundColor(Color backgroundColor)</code> Sets the background color of this WebpageObject.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setBuildParameters(BuildParameters parameters)</code> Sets the Buildparameters for this WebpageObject</td>
</tr>
<tr>
<td>public void</td>
<td><code>setForbiddenPosition(boolean forbiddenPosition)</code> Sets whether or not the WebpageObject lies on a forbidden position.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setMoveColor(Color moveColor)</code> The color of the border when the WebpageObject is moving!</td>
</tr>
<tr>
<td>public void</td>
<td><code>setMoving(boolean moving)</code> Sets whether or not the WebpageObject is moving.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setPlaced(boolean placed)</code> Sets whether or not the WebpageObject is placed on the Webpage already.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setPropertyToolbar(JPanel propertyToolbar)</code> Sets the property-toolbar for this specific WebpageObject.</td>
</tr>
<tr>
<td>public void</td>
<td><code>setSelectColor(Color selectColor)</code> Sets the Color of the shade shown when this color is selected!</td>
</tr>
<tr>
<td>public void</td>
<td><code>setSelected(boolean selected)</code> Sets whether or not this WebpageObject is selected.</td>
</tr>
</tbody>
</table>
### C.3. LAYOUT.WEBPAGE

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>setSelectStroke(BasicStroke selectStroke) Sets the stroke to draw the border when the WebpageObject is selected.</td>
</tr>
<tr>
<td>public void</td>
<td>setUnselectColor(Color unselectColor) Sets the color for the border if the WebpageObject is not selected.</td>
</tr>
<tr>
<td>public String</td>
<td>transformXSLStylesheet(String xml) Transforms the XML-representation of the given WebpageObject into the XSL-template needed to create the website.</td>
</tr>
</tbody>
</table>

### C.3.16 layout.webpage.WebpageObjectFactory

java.lang.Object

public WebpageObjectFactory extends Object

This is a factory class which is used to produce an instance of a subtype of the interface WebpageObject

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageComponent</td>
<td>createWebpageComponent(WebpageContainer parent, CMComponent component, GUIObject obj) Creates a WebpageComponent that represents the given CMComponent</td>
</tr>
<tr>
<td>public WebpageContainer</td>
<td>createWebpageContainer(WebpageContainer parent, CMContainer container) Creates a WebpageContainer that represents the given CMContainer</td>
</tr>
</tbody>
</table>

### C.3.17 layout.webpage.WebpageObjectPopupActionHandler

java.lang.Object

public WebpageObjectPopupActionHandler extends Object
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

implements ActionListener, PopupActionHandler

This class shows a popup-menu when the popup-Trigger is pressed on a specific WebpageObject!

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageObjectPopupActionHandler(WebpageObject webpageObject, WebpageModel webpageModel) Creates a new instance of WebpageObjectPopupActionHandler</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>actionPerformed(ActionEvent e) Invoked when a menuitem on the popup menu is selected. Calls the desired methods.</td>
</tr>
<tr>
<td>public void</td>
<td>setPageamodel(WebpageModel webpageModel) In certain cases (e.g. the webpage) the webpageModel is not known when the constructor is invoked. In this case the model can be set with this method.</td>
</tr>
<tr>
<td>public void</td>
<td>showPopupMenu(Point p) This method creates a popup menu for the selected content and shows it on the given coordinate.</td>
</tr>
</tbody>
</table>

C.3.18 layout.webpage.WebpagePictureComponent

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
layout.webpage.WebpageComponent

public WebpagePictureComponent
extends WebpageComponent

This class represents a CMPicture type. It shows a picture on the screen. To ensure this the ResourceManager must return the correct path to the picture.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpagePictureComponent(CMPicture cmPicture, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor) Creates a new instance of WebpageTextComponent</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><strong>calculateSize</strong>(int width) Calculates the size of this WebpagePictureComponent. This method only returns the size of the picture it contains. The width parameter is not used!</td>
</tr>
<tr>
<td>public Element</td>
<td><strong>createXMLLayout</strong>() This method creates the XML-Element which stores the layout of the CMPicture represented by this WebpageComponent! With this XML-Element a XSL-template generates then the template to render the CMPicture on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent</strong>() Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent</strong>(Dimension size, Point location) Draws this component on the page. This Method is responsible to set position and size of the component and to get its content!</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getLayoutTemplateFileName</strong>() Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject</strong>(Element webpageObjectXMLTemplate) Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><strong>relocateContent</strong>(Point newLocation, Dimension newSize) Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content.</td>
</tr>
</tbody>
</table>

### C.3.19 layout.webpage.WebpagePictureLinkComponent

```java
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
layout.webpage.WebpageComponent
```

```java
public WebpagePictureLinkComponent extends WebpageComponent
```

This class represents a CMLink which has a picture as a resource. It is necessary to distinguish between links with pictures as resources and links with a text because they are extracted from the db and displayed on the screen differently.

### Constructors
### APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WebpagePictureLinkComponent</strong>(CMLink cmLink, GUIObject resourceObject, GUIObject urlObject, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor) Creates a new instance of WebpagePictureLinkComponent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><code>calculateSize</code>(int width) Calculates the size of this CMPictureLinkComponent. This method just returns the picture’s size. The parameter width is ignored.</td>
</tr>
<tr>
<td>public Element</td>
<td><code>createXMLLayout</code>() This method creates the XML-Element which stores the layout of the CMLink represented by this WebpageComponent! With this XML-Element a XSL-stylesheet then generates the template to render the CMLink on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent</code>() Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td><code>drawContent</code>(Dimension size, Point location) Draws this component on the page. This Method is responsible to set position and size of the component and to get its content!</td>
</tr>
<tr>
<td>public String</td>
<td><code>getLayoutTemplateFileName</code>() Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public void</td>
<td><code>loadWebpageObject</code>(Element webpageObjectXMLTemplate) Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>relocateContent</code>(Point newLocation, Dimension newSize) Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content.</td>
</tr>
<tr>
<td>public void</td>
<td><code>saveToDatabase</code>(Element template, String templateName) Writes the given template to the database and associates it with the given WebpageObject. This method is called for all WebpageObjects except on of the type Webpage. For Webpage objects, there is a special method <code>saveWebpageToDatabase()</code>. I must subclass the normally used method of the superclass WebpageComponent because CMLink has two child-objects (resource and URL) to save as well!</td>
</tr>
</tbody>
</table>
C.3.20 layout.webpage.WebpageSitemapComponent

This class represents a CMSitemap. As the sitemap is created by the CMServer at the moment of the creation of the website we show a dummy placeholder. This has no influence on the functionality of the rendering process which works absolutely fine.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageSitemapComponent(CMSitemap cmSitemap, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor) Creates a new instance of WebpageTextComponent</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td>calculateSize(int width) Calculate the size of the component with the given width. This Method is needed to set the size of the parentContainer correctly. In this special case we just return a default size</td>
</tr>
<tr>
<td>public Element</td>
<td>createXMLLayout() This method creates the XML-Element which stores the layout of the CMNavigation represented by this WebpageComponent! With this XML-Element a XSL-stylesheet generates the template to render the CMSitemap on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td>drawContent() Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td>drawContent(Dimension size, Point location) Draws this component on the page.</td>
</tr>
<tr>
<td>public String</td>
<td>getContent() Gets the content of this CMNavigation</td>
</tr>
<tr>
<td>public Color</td>
<td>getFontColor() The font color</td>
</tr>
<tr>
<td>public String</td>
<td>getLayoutTemplateFileName() Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
</tbody>
</table>
### Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject</strong>&lt;br&gt;Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><strong>relocateContent</strong>&lt;br&gt;Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content. It the case of WebpageNavigationComponent the new minimum size of the content is calculated. A check for intersections follows. If there are any, false is returned. Otherwise, the text is reformatted, then content is drawn and finally the method returns true.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setFontColor</strong>&lt;br&gt;Sets the font color</td>
</tr>
</tbody>
</table>

### C.3.21 `layout.webpage.WebpageTextComponent`

- `java.lang.Object`
- `java.awt.Component`
- `java.awt.Container`
- `javax.swing.JComponent`
- `javax.swing.JLabel`
- `layout.webpage.WebpageComponent`

```java
public WebpageTextComponent
extends WebpageComponent
```

This class represents the CMText type in the CMServer data model. It shows the text on the screen and allows the changing font, font style, font size, fontcolor

#### Constructors

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WebpageTextComponent</strong>&lt;br&gt;(CMText cmText, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor)</td>
<td>Creates a new instance of WebpageTextComponent</td>
<td></td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><strong>calculateSize</strong>(int width) Calculate the size of the component with the given width. This Method is needed to set the size of the parentContainer correctly.</td>
</tr>
</tbody>
</table>
C.3. LAYOUT.WEBPAGE

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Element</td>
<td>createXMLLayout() This method creates the XML-Element which stores the layout of the CMText represented by this WebpageComponent! With this XML-Element a XSL-stylesheet generates the template to render the CMText on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td>drawContent() Draws this component on the page. This Method assumes that the size and the position of the component are already defined.</td>
</tr>
<tr>
<td>public void</td>
<td>drawContent(Dimension size, Point location) Draws this component on the page. This means creating the text and applying the correct font. Then the JLabel does the rest.</td>
</tr>
<tr>
<td>public String</td>
<td>getContent() Gets the content of this CMNavigation</td>
</tr>
<tr>
<td>public Color</td>
<td>getFontColor() Gets the font color</td>
</tr>
<tr>
<td>public String</td>
<td>getLayoutTemplateFileName() Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public Font</td>
<td>getTextFont() Returns the used font</td>
</tr>
<tr>
<td>public void</td>
<td>loadWebpageObject(Element webpageObjectXML-Template) Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td>relocateContent(Point newLocation, Dimension newSize) Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content. It the case of WebpageTextComponent I calculate the new minimum size of the content. Then I check for intersections. If there are any, false is returned. Otherwise, the text is reformatted, then content is drawn and finally the method returns true.</td>
</tr>
<tr>
<td>public void</td>
<td>setFontColor(Color fontColor) Sets the font color</td>
</tr>
<tr>
<td>public void</td>
<td>setTextFont(Font f) Sets the font of this CMNavigation.</td>
</tr>
</tbody>
</table>

C.3.22 layout.webpage.WebpageTextLinkComponent

```
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JLabel
layout.webpage.WebpageComponent
```

public WebpageTextLinkComponent
extends WebpageComponent

This class represents a CMLink which has a text as a resource. It is necessary to distinguish
between links with pictures as resources and links with a text because they are extracted from the db and displayed on the screen differently.

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WebpageTextLinkComponent</strong>&lt;br&gt;(CMLink cmLink, GUIObject resourceObject, GUIObject urlObject, WebpageContainer container, BuildParameters parameters, ResourceManager resourceManager, Color selectColor, Color backgroundColor, Color moveColor, Color unselectColor)</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Dimension</td>
<td><strong>calculateSize</strong>(int width) Calculate the size of the component with the given width. This Method is needed to set the size of the parentContainer correctly.</td>
</tr>
<tr>
<td>public Element</td>
<td><strong>createXMLLayout()</strong> This method creates the XML-Element which stores the layout of the CMLink represented by this WebpageComponent! With this XML-Element a XSL-stylesheets generates the template to render the CMLink on the webpage!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent()</strong> Draws this component on the page. This Method assumes that the size and the position of the component are already defined. the component and to get its content!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>drawContent</strong>(Dimension size, Point location) Draws this component on the page.</td>
</tr>
<tr>
<td>public CMComponent</td>
<td><strong>getCMResource()</strong> Gets the resource of this CMLink.</td>
</tr>
<tr>
<td>public CMUrl</td>
<td><strong>getCMUrl()</strong> Gets the URL of this CMLink.</td>
</tr>
<tr>
<td>public String</td>
<td><strong>getLayoutTemplateFileName()</strong> Returns the name of the XSL-stylesheet needed to generate the template used to create the html code for of this WebpageObject.</td>
</tr>
<tr>
<td>public Color</td>
<td><strong>getLinkColor()</strong> Gets the color of the link.</td>
</tr>
<tr>
<td>public Font</td>
<td><strong>getTextFont()</strong> Gets the used font.</td>
</tr>
<tr>
<td>public void</td>
<td><strong>loadWebpageObject</strong>(Element webpageObjectXMLTemplate) Loads the properties of this WebpageObject from the given Element</td>
</tr>
<tr>
<td>public boolean</td>
<td><strong>relocateContent</strong>(Point newLocation, Dimension newSize) Relocates and resizes the content to the given location and size. Returns true if it is possible to resize the content. It the case of WebpageTextComponent I calculate the new minimum size of the content. Then I check for intersections. If there are any, false is returned. Otherwise, the text is reformatted, then content is drawn and finally the method returns true.</td>
</tr>
</tbody>
</table>
C.4. LAYOUT.DND

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>saveToDatabase(Element template, String template-Name) Writes the given template to the database and associates it with the given WebpageObject. This method is called for all WebpageObjects except on of the type Webpage. For Webpage objects, there is a special method saveWebpageToDatabase()</td>
</tr>
<tr>
<td>public void</td>
<td>setLinkColor(Color linkColor) Sets the color of the link.</td>
</tr>
<tr>
<td>public void</td>
<td>setTextFont(Font f) Sets the font used.</td>
</tr>
</tbody>
</table>

C.4 layout.dnd

C.4.1 layout.dnd.TransferableWebpageContainer

java.lang.Object

public TransferableWebpageContainer
extends Object
implements Transferable

This class implements the Transferable interface. As we want to transfer data only within the same process (even the same thread). This implementation is not very difficult. It only holds the name of the WebpageObject we want to place on the screen.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransferableWebpageContainer(String webpageContainerName) Creates a new instance of TransferableWebpageContainer</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Object</td>
<td>getTransferData(DataFlavor flavor) Returns an object which represents the data to be transferred. The class of the object returned is defined by the representation class of the flavor.</td>
</tr>
<tr>
<td>public DataFlavor[]</td>
<td>getTransferDataFlavors() Returns an array of DataFlavor objects indicating the flavors the data can be provided in. The array should be ordered according to preference for providing the data (from most richly descriptive to least descriptive).</td>
</tr>
<tr>
<td>public boolean</td>
<td>isDataFlavorSupported(DataFlavor flavor) Returns whether or not the specified data flavor is supported for this object.</td>
</tr>
</tbody>
</table>
C.4.2  layout.dnd.TreeDragGestureListener

    java.lang.Object

public TreeDragGestureListener
extends Object
implements DragGestureListener

This class invokes DragGestureListener. It is added to the DesignGUITree and tracks the
mouse to observe if the user makes a drag-gesture. If so it invokes the drag-action.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeDragGestureListener(DesignGUITree designTree, DragSource dragSource, Tree-</td>
</tr>
<tr>
<td>DragSourceListener treeDragSourceListener, WebpageModel webpageModel) Creates</td>
</tr>
<tr>
<td>a new instance of TreeDragGestureListener</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>dragGestureRecognized(DragGestureListener e) Invoked when the drag-gesture</td>
</tr>
<tr>
<td></td>
<td>occurs. This method gets the name of currently selected GUIObject. It</td>
</tr>
<tr>
<td></td>
<td>passes this name to the Transferable and then starts the drag-action.</td>
</tr>
</tbody>
</table>

C.4.3  layout.dnd.TreeDragSourceListener

    java.lang.Object
    java.awt.dnd.DragSourceAdapter

public TreeDragSourceListener
extends DragSourceAdapter

This class implements the DragSourceListener interface. Its only job is to change the mouse
cursor when the mouse moves over a object enabled to be dragged!

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeDragSourceListener() Creates a new instance of TreeDragSourceListener</td>
</tr>
</tbody>
</table>

Methods
C.4. LAYOUT.DND

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>dragEnter(DragSourceDragEvent e) Invoked when the mouse enters the region which is enabled for dragging.</td>
</tr>
</tbody>
</table>

C.4.4 layout.dnd.WebpageDropTargetListener

java.lang.Object
java.awt.dnd.DropTargetAdapter

public WebpageDropTargetListener
extends DropTargetAdapter

This class implements the DropTargetListener interface. It is added to a Webpage object to enable the webpage to receive drop-data. Furthermore, during the drag-action it shows a rectangular shape of the object currently dragged to give the user feedback on the current size and position.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebpageDropTargetListener(Webpage webpage, WebpageModel webpageModel, DesignGUITree designTree, ResourceManager resourceManager) Creates a new instance of WebpageDropTargetListener</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>dragEnter(DropTargetDragEvent e) Invoked when the mouse enters the drop-region. Calls drawRectangle to draw the rectangle with the shape of the WebpageObject to be dropped.</td>
</tr>
<tr>
<td>public void</td>
<td>dragExit(DropTargetEvent e) Invoked when the mouse exits the drop-region. Makes the rectangle showing the WebpageObject to be dropped invisible.</td>
</tr>
<tr>
<td>public void</td>
<td>dragOver(DropTargetDragEvent e) Invoked when the mouse moves over the drop-region. Calls drawRectangle to draw the rectangle with the shape of the WebpageObject to be dropped.</td>
</tr>
</tbody>
</table>
public void drop(DropTargetDropEvent e) Called when the drag operation has terminated with a drop on the operable part of the drop site for the DropTarget registered with this listener. This method is responsible for undertaking the transfer of the data associated with the gesture. The DropTargetDropEvent provides a means to obtain a Transferable object that represents the data object(s) to be transferred.

From this method, the DropTargetListener shall accept or reject the drop via the acceptDrop(int dropAction) or rejectDrop() methods of the DropTargetDropEvent parameter.

Subsequent to acceptDrop(), but not before, DropTargetDropEvent’s getTransferable() method may be invoked, and data transfer may be performed via the returned Transferable’s getTransferData() method.

At the completion of a drop, an implementation of this method is required to signal the success/failure of the drop by passing an appropriate boolean to the DropTargetDropEvent’s dropComplete(boolean success) method.

++ Note: The data transfer should be completed before the call to the DropTargetDropEvent’s dropComplete(boolean success) method. After that, a call to the getTransferData() method of the Transferable returned by DropTargetDropEvent.getTransferable() is guaranteed to succeed only if the data transfer is local; that is, only if DropTargetDropEvent.isLocalTransfer() returns true. Otherwise, the behavior of the call is implementation-dependent.

public void setWebpageModel(WebpageModel webpageModel)
Sets the WebpageModel which manages the webpage

### C.5 layout.stylesheet

#### C.5.1 layout.stylesheet.XMLLayoutGenerator

java.lang.Object

public XMLLayoutGenerator extends Object

This class takes a WebpageContainer and generates a xml-representation of the layout as it is at the moment. The XML document satisfies the DTD WebpageLayout.dtd defined in
the package An XSL-File is then applied to the generated XML-File to create a stylesheet which constructs the layout of the container out of the data in the database. This stylesheet is also produced during the execution of this class.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLLayoutGenerator(WebpageContainer webpageContainer) Creates a new instance of XMLLayoutGenerator</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void createFile()</td>
<td>This simple creates a XML file containing the layout of the WebpageContainer. The method is mainly used for debugging.</td>
</tr>
<tr>
<td>public Element</td>
<td>getLayoutedContainer() Returns the jdom element containing the container’s layout.</td>
</tr>
<tr>
<td>public Document</td>
<td>getLayoutedContainerDocument() Returns a jdom document containing the container’s layout.</td>
</tr>
</tbody>
</table>

C.5.2 layout.stylesheet.XMLLayoutLoader

java.lang.Object

public XMLLayoutLoader
extends Object

This class takes a webpage object and then loads the layout as it is specified in the XML layout file associated with the CMContainer representing the webpage.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLLayoutLoader(Webpage webpage) Creates a new instance of XMLLayoutGenerator</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void loadWebpage()</td>
<td>This method must be called to start the loading of the layout. It just gets the correct XML layout file then it invokes the methods loadContainer and loadComponent to do the actual work.</td>
</tr>
</tbody>
</table>
C.6 layout.properties

C.6.1 layout.properties.CMContainerPropertyToolbar

```java
public CMContainerPropertyToolbar
extends JPanel
implements PropertyToolbar
```

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webPageObject() Sets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>setVisible(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>setWebpageObject(WebpageObject webpageContainer) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.2 layout.properties.CMDefaultPropertyToolbar

```java
public CMDefaultPropertyToolbar
extends JPanel
implements PropertyToolbar
```
A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

### Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMDefaultPropertyToolbar</strong>(WebpageObject webpageObject, ResourceManager resourceManager)</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td><strong>getWebpageObject()</strong> Sets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setVisible(boolean visible)</strong> This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setWebpageObject(WebpageObject webpageObject)</strong> Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

### C.6.3 layout.properties.CMMixedPropertyToolbar

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject.
Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMMixedPropertyToolbar(WebpageMixedContainer webpageMixedContainer, ResourceManager resourceManager) Creates a new CMMixedPropertiesToolbar</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject() Sets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>public void</td>
<td>setVisible(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmer's responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>public void</td>
<td>setWebpageObject(WebpageObject webpageMixed-TextComponent) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.4 layout.properties.CMNavigationPropertyToolbar

```java
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel
```

```java
public CMNavigationPropertyToolbar
extends JPanel
implements PropertyToolbar
```

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors
C.6. LAYOUT.PROPERTIES

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMNavigationPropertyToolbar</strong> (WebpageNavigationComponent webpageNavigationComponent, ResourceManager resourceManager) Creates a new CMNavigationPropertyToolbar</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td><strong>getWebpageObject()</strong> Sets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setVisible(boolean visible)</strong> This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar <strong>must be updated before it is displayed again</strong>!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setWebpageObject(WebpageObject webpageNavigationComponent)</strong> Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.5 layout.properties.CMPictureLinkPropertyToolbar

```java
java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel
```

```java
public CMPictureLinkPropertyToolbar extends JPanel implements PropertyToolbar
```

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

**Constructors**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMPictureLinkPropertyToolbar</strong> (WebpagePictureLinkComponent webpagePictureLinkComponent, ResourceManager resourceManager) Creates a new CMPictureLinkPropertyToolbar</td>
</tr>
</tbody>
</table>
APPENDIX C. THE API OF THE GRAPHICAL TEMPLATE EDITOR

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject() Sets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>public void</td>
<td>setVisible(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObject’s background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>public void</td>
<td>setWebpageObject(WebpageObject webpagePictureLinkComponent) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.6 layout.properties.CMPicturePropertyToolbar

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel

public CMPicturePropertyToolbar extends JPanel
implements PropertyToolbar

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPicturePropertyToolbar(WebpagePictureComponent webpagePictureComponent, ResourceManager resourceManager) Creates a new CMPicturePropertyToolbar</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject() Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>
C.6. LAYOUT.PROPERTIES

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>setVisible(boolean visible)  This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>public void</td>
<td>setWebpageObject(WebpageObject webpagePicture-Component) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.7 layout.properties.CMSitemapPropertyToolbar

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel

public CMSitemapPropertyToolbar extends JPanel
implements PropertyToolbar

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSitemapPropertyToolbar(WebpageSitemapComponent webpageSitemapComponent, ResourceManager resourceManager) Creates a new CMSitemapPropertyToolbar</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject() Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>
Returns | Description
---|---
public void | setVisible(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!

public void | setWebpageObject(WebpageObject webpage-SitemapComponent) Sets the Content for which the toolbar is responsible

C.6.8 `layout.properties.CMTextLinkPropertyToolbar`

```java
java.lang.Object
doesnt implement Component Container JComponent JComponent JPanel
```

public CMTextLinkPropertyToolbar extends JPanel implements PropertyToolbar

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>CMTextLinkPropertyToolbar(WebpageTextLinkComponent webpageTextLinkComponent, ResourceManager resourceManager)</code> Creates a new CMTextLinkPropertyToolbar</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td><code>getWebpageObject()</code> Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>
C.6. LAYOUT.PROPERTIES

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td><strong>setVisible</strong>(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar <strong>must be updated before it is displayed again</strong>!</td>
</tr>
<tr>
<td>public void</td>
<td><strong>setWebpageObject</strong>(WebpageObject webpageTextLinkComponent) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>

C.6.9 layout.properties.CMTextPropertyToolbar

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JPanel

public CMTextPropertyToolbar extends JPanel implements PropertyToolbar

A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

Constructors

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMTextPropertyToolbar(WebpageTextComponent webpageTextComponent, ResourceManager resourceManager) Creates a new CMTextLinkPropertyToolbar</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td><strong>getWebpageObject()</strong> Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>
### C.6.10 layout.properties.ColorchooserDialog

```java
public interface ColorchooserDialog
```

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Color</td>
<td><code>getSelectedColor()</code> Gets the SelectedColor</td>
</tr>
<tr>
<td>public boolean</td>
<td><code>isCanceled()</code> Checks wether or not the cancel button was clicked.</td>
</tr>
</tbody>
</table>

#### Constructors

```java
ColorchooserDialog(JFrame parent, boolean modal, ResourceManager resourceManager)
```

Simple Dialog to choose a Color. Used in different Property-toolbars.

#### Methods

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ColorchooserDialog(JFrame parent, boolean modal, ResourceManager resourceManager)</strong></td>
<td>Creates new form ColorchooserDialog</td>
</tr>
</tbody>
</table>

### C.6.11 layout.properties.PropertyToolbar

```java
public interface PropertyToolbar
```

public interface *PropertyToolbar*
A PropertyToolbar allows the user to modify the properties of a specific WebpageObject. Properties are things like: Font, background color etc. For each WebpageObject there must be a specific subclass of this interface. This interface defines all methods each of the property-toolbars for the different CMComponents must implement.

### Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public WebpageObject</td>
<td>getWebpageObject() Gets the Content for which the toolbar is responsible</td>
</tr>
<tr>
<td>public void</td>
<td>setVisible(boolean visible) This method must be called when the property toolbar must be shown or hidden. If it must be shown, then its the programmers responsibility to adjust the toolbar to the latest changes of the properties of its WebpageObject. E.g. when the WebpageObjects background color is changed. Short: The PropertyToolbar must be updated before it is displayed again!</td>
</tr>
<tr>
<td>public void</td>
<td>setWebpageObject(WebpageObject webpageObject) Sets the Content for which the toolbar is responsible</td>
</tr>
</tbody>
</table>
Appendix D

The Stylesheets

This appendix contains the DTD for the XML representation and the code of stylesheets used to create the templates.
D.1 DTD

<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document : Webpage.dtd
  Created on : 3. Januar 2003, 18:06
  Author : Patrick Buechler
  Description:
  This DTD defines how the layout of a webpage object and its components is represented as an XML-Document
-->

<!ELEMENT webpage (layout,fontlist)>
<!ATTLIST webpage
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  fixedWidth CDATA #IMPLIED
>
<!ELEMENT layout (row+)>
<!ATTLIST layout
  width CDATA #REQUIRED
  height CDATA #REQUIRED
>
<!ELEMENT row (column+)>  
<!ATTLIST row
  width CDATA #IMPLIED
>
<!ELEMENT column(spacer|collection|mixed|text|mixedText|link|picture
|navigation|sitemap|defaultcomponent|defaultcontainer)>  
<!ATTLIST column
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  colspan CDATA #REQUIRED
  rowspan CDATA #REQUIRED
>
<!ELEMENT collection (layout)>  
<!ATTLIST collection
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED
>
<!ELEMENT mixed (mixedText, layout)>
<!ATTLIST mixed
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>

<!ELEMENT text (font?)>
<!ATTLIST text
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED>

<!ELEMENT mixedText (font?)>
<!ATTLIST mixedText
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>

<!ELEMENT link (collection|mixed|text|link|picture
defaultcomponent|defaultcontainer)>
<!ATTLIST link
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  parent_id CDATA #REQUIRED
  backgroundColor CDATA #REQUIRED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>
D.1. DTD

```xml
<!ELEMENT fontlist (font*)>
<!ELEMENT spacer EMPTY>
  <!ATTLIST spacer
    path CDATA #REQUIRED
    backgroundColor CDATA #REQUIRED
  >
<!ELEMENT font EMPTY>
  <!ATTLIST font
    font_id CDATA #REQUIRED
    name CDATA #REQUIRED
    style (plain|bold|italic|bolditalic) #REQUIRED
    size CDATA #REQUIRED
    color CDATA #REQUIRED
  >
```
D.2 Webpage

<!--The template for the whole webpage -->
<xsl:template match="/webpage">
  <!--template rule matching source root element -->
  &lt;xsl:text disable-output-escaping="yes">
  &lt;xsl:template match="/">
  <!--lets start with the HTML-->
  <html>
    <head>
      <title>
        &lt;xsl:text disable-output-escaping="yes">
        &lt;xsl:value-of select="/webobject/property[@name='title']/string" />
        &lt;/xsl:text>
      </title>
      <style type="text/css">
        &lt;xsl:apply-templates select="/webpage/fontlist"/>
      </style>
    </head>
    <body>
      <xsl:attribute name="bgcolor">
        &lt;xsl:value-of select="@backgroundColor"/>
      </xsl:attribute>
      <!--Lets load the layout, we must pass the id of this container to ensure only its children are selected-->
      &lt;xsl:apply-templates select="/webpage/layout"&gt;
        &lt;xsl:with-param name="parent_id">
          &lt;xsl:value-of select="@oid"/>
        &lt;/xsl:with-param>
        &lt;xsl:with-param name="backgroundColor">
          &lt;xsl:value-of select="@backgroundColor"/>
        &lt;/xsl:with-param>
      &lt;/xsl:apply-templates>
    </body>
  </html>
&lt;xsl:text disable-output-escaping="yes">
&lt;xsl:template>

<!--This template creates the fontlist in the header of the file-->
<xsl:template match="fontlist">
  &lt;xsl:for-each select="child::*">
    &lt;xsl:value-of select="attribute::font_id"/>
  </xsl:for-each>
</xsl:template>
D.2. WEBPAGE

```xml
<xsl:template>
  <xsl:for-each>
    <xsl:choose>
      <xsl:when test="attribute::style='plain'">
        <xsl:text>{color:
          font-family:
          font-size:
        }
      </xsl:when>
      <xsl:when test="attribute::style='bold'">
        <xsl:text>font-weight:bold} }
      </xsl:when>
      <xsl:when test="attribute::style='italic'">
        <xsl:text>font-style:italic} }
      </xsl:when>
      <xsl:when test="attribute::style='bolditalic'">
        <xsl:text>font-style:italic; font-weight:bold} }
      </xsl:when>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>
```
D.3 Layout

<!--This template applies to the layout element, which is the base element to manage the layout of a page or a container -->
<xsl:template match="layout">
  <xsl:param name="parent_id">0</xsl:param>
  <xsl:param name="backgroundColor">white</xsl:param>
  <table>
    <xsl:attribute name="width"><xsl:value-of select="@width"/></xsl:attribute>
    <xsl:attribute name="bgcolor"><xsl:value-of select="$backgroundColor"/></xsl:attribute>
    <!--Les create the columns of this row-->
    <xsl:apply-templates select="row">
      <xsl:with-param name="parent_id"><xsl:value-of select="$parent_id"/></xsl:with-param>
    </xsl:apply-templates>
  </table>
</xsl:template>

D.4 Row

<!--This template applies to a row in a layout, it determines its columns and then passes the control to the columns-template -->
<xsl:template match="row">
  <xsl:param name="parent_id">0</xsl:param>
  <tr>
    <xsl:apply-templates select="column">
      <xsl:with-param name="parent_id"><xsl:value-of select="\$parent_id"/></xsl:with-param>
    </xsl:apply-templates>
  </tr>
</xsl:template>
D.5 Column

<!--This template applies to a column in a layout, it determines its size and content-->
<xsl:template match="column">
  <xsl:param name="parent_id">0</xsl:param>
  <td>
    <xsl:attribute name="colspan">
      <xsl:value-of select="@colspan"/>
    </xsl:attribute>
    <xsl:attribute name="rowspan">
      <xsl:value-of select="@rowspan"/>
    </xsl:attribute>
    <xsl:attribute name="width">
      <xsl:value-of select="@width"/>
    </xsl:attribute>
    <xsl:attribute name="height">
      <xsl:value-of select="@height"/>
    </xsl:attribute>
    <xsl:attribute name="valign">
      top
    </xsl:attribute>
    <xsl:for-each select="child::*">
      <xsl:attribute name="bgcolor">
        <xsl:value-of select="attribute::backgroundColor"/>
      </xsl:attribute>
    </xsl:for-each>
    <xsl:choose>
      <!-- lets a cmixed-text-template-->
      <xsl:when test="mixedText">
        <span>
          <xsl:attribute name="class">
            <xsl:value-of select="/font/attribute::font_id"/>
          </xsl:attribute>
          <xsl:text disable-output-escaping="yes">
            <xsl:value-of select="/components/webobject[@oid='
            <xsl:value-of select="$parent_id"/>
            '/property[@name='content']/mixed" />
          </xsl:text>
        </span>
      </xsl:when>
      <!-- lets call the template to load a spacer image because there is no child-->
      <xsl:when test="spacer">
        <xsl:text disable-output-escaping="yes">
          &lt;xsl:text disable-output-escaping="yes">
            
          </xsl:text>
        </xsl:text>
      </xsl:when>
    </xsl:choose>
  </td>
</xsl:template>
D.6 Collection

<!--This is the template snippet responsible to create the correct rendering of a cmcollection.-->
<xsl:template match="collection">
  <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called-->  
  <xsl:text disable-output-escaping="yes"> <!--template rule matching a cmcollection-->
    &lt; !--template rule matching a cmcollection-->
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    &lt; xsl:template
      match=&quot;/webobject[@oid='"]<xsl:with-param name="parent_id">
        <xsl:value-of select="@oid"/>
        '/webobject[@oid='"
      ]&quot; />
      &lt;xsl:with-param name="backgroundColor">
        <xsl:value-of select="@backgroundColor"/>
        &lt;xsl:with-param>
          <xsl:value-of select="@backgroundColor"/>
        </xsl:with-param>
        &lt;xsl:apply-templates select="layout">
          <xsl:with-param name="parent_id">
            <xsl:value-of select="@oid"/>
          </xsl:with-param>
          <xsl:with-param name="backgroundColor">
            <xsl:value-of select="@backgroundColor"/>
          </xsl:with-param>
          &lt;xsl:apply-templates>
            &lt;xsl:text disable-output-escaping="yes">
              &lt; /xsl:template>
            </xsl:text>
          </xsl:apply-templates>
        </xsl:with-param>
        &lt;xsl:apply-templates>
          &lt;xsl:text disable-output-escaping="yes">
            &lt; /xsl:template>
          </xsl:text>
        </xsl:apply-templates>
      </xsl:template>
    </xsl:text>
D.7 Mixed

<!--This is the template snippet responsible to create the correct rendering of a cmmixed.-->
<xsl:template match="mixed">
  <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called -->
  &lt; !--template rule matching a cmmixed-->
</xsl:template>

<xsl:template disable-output-escaping="yes">
  &lt; xsl:template match="//webobject[@oid=' &lt;xsl:value-of select="@parent_id"/]">//webobject[@oid=' &lt;xsl:value-of select="@oid"/>
  &lt;xsl:text disable-output-escaping="yes" />
  &lt;\"/>
</xsl:template>

<!--Lets load the layout, we must pass the id of this container to ensure only its children are selected-->
<xsl:apply-templates select="layout">
  <xsl:with-param name="parent_id">
    <xsl:value-of select="@oid"/>
  </xsl:with-param>
  <xsl:with-param name="backgroundColor">
    <xsl:value-of select="@backgroundColor"/>
  </xsl:with-param>
</xsl:apply-templates>

</xsl:template>
D.8 Text

<!--This is the template snippet responsible to create the correct rendering of a cmtext.-->
<xsl:template match="text">
    <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called -->
    <xsl:template disable-output-escaping="yes">
        &lt;!--template rule matching a cmtext-->  
        </xsl:template>
        <xsl:template disable-output-escaping="yes">
            &lt; xsl:template match="//webobject[@oid=' 
            </xsl:template>
            <xsl:value-of select="@parent_id"/>
            <xsl:template disable-output-escaping="yes">
                ']//webobject[@oid=' 
                </xsl:template>
                <xsl:value-of select="@oid"/>
                <xsl:template disable-output-escaping="yes">
                    ']&quot;  >
                </xsl:template>
            </xsl:template>
        <span>
            <xsl:attribute name="class">
                <xsl:value-of 
                select="child::*/attribute::font_id"/>
            </xsl:attribute>
            <!--Lets get the text--> 
            <xsl:template disable-output-escaping="yes">
                &lt;xsl:value-of select="&quot; property[@name='content']/string&quot;/>
            </xsl:template>
        </span>
    </xsl:template>
</xsl:template>
D.9 URL

<!--This is the template snippet responsible to create the correct rendering of a cmurl.-->  
<xsl:template match="url">
  <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called -->
  <xsl:text disable-output-escaping="yes"> <!--template rule matching a cmurl-->
    &lt; !--template rule matching a cmurl-->
  </xsl:template>
</xsl:template>
D.10 Picture

<!--This is the template snippet responsible to create the correct rendering of a cmpicture.-->
<xsl:template match="picture">
  <!--First we create the Template-header
       with the correct oid, name and parent_id
       to make sure only the correct stylesheets are called -->
  <!--template rule matching a cmpicture-->
  <xsl:text disable-output-escaping="yes">
    <!--template rule matching a cmpicture-->
  </xsl:text>
  <xsl:template match="//webobject[@oid='"$
    <xsl:value-of select="@parent_id"/>
    "']///webobject[@oid=""
      <xsl:value-of select="@oid"/>
    "']>
    <!--Lets get the image-path and load the image,
        in case the path is not found we print the path
        to help find the cause of the problem. This is
        a bit complicated, because at the moment the
        XALAN-processor creates unclosed element-tags
        when the elementbody is empty and the output-type
        is xhtml. Therefore I try to pass the img-element as
        xsl:variable select="property[@name='image']/string"/>
    </xsl:text>
    <xsl:text disable-output-escaping="yes">
      <img src="$
        <xsl:value-of select="$imgPath"/>
      "/>
    </xsl:text>
  </xsl:template>
</xsl:template>
APPENDIX D. THE STYLESHEETS

```
&amp;quot; alt=&amp;quot;picture not found, path: 
&lt;/xsl:text&gt;
&lt;xsl:text disable-output-escaping="yes">
 &lt;/xsl:text&gt;
&lt;xsl:text disable-output-escaping="yes">
 &lt;xsl:value-of select=&amp;quot;
 \$imgPath&amp;quot;/&gt;
&lt;/xsl:text&gt;
&lt;xsl:text disable-output-escaping="yes">
 &lt;xsl:text disable-output-escaping=
&amp;quot;yes&amp;quot;&gt;
&lt;/xsl:text&gt;
&lt;xsl:text disable-output-escaping="no">
 &quot; /&gt;
&lt;/xsl:text&gt;
&lt;xsl:text disable-output-escaping="yes">
 &lt;/xsl:text&gt;
&lt;xsl:text&gt;
 &lt;xsl:text disable-output-escaping="yes">
 &lt; /xsl:template&gt;
&lt;/xsl:text&gt;
&lt;/xsl:template&gt;
```
D.11 Sitemap

<!--This is the template snippet responsible to create the
correct rendering of a cmsitemap.-->
<xsl:template match="sitemap">
  <xsl:text disable-output-escaping="yes">
    <!--template rule matching a default cmsitemap-->
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    <!-- template for a sitemap component -->
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    <xsl:template>
      <table border="0" cellpadding="5" cellspacing="5" width="100%">
        <tr>
          <td>
            <h2>
              <xsl:text disable-output-escaping="yes"
                ><xsl:value-of select="property[@name='title']/string"/>
            </h2>
            <dl>
              <xsl:text disable-output-escaping="yes">
                <xsl:apply-templates
                  mode="link"
                  select="components"/>
              </xsl:text>
            </dl>
          </td>
        </tr>
      </table>
    </xsl:template>
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    <!-- template for a directory component -->
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    <xsl:template
      match="webobject[@type='cmdirectory']"
      mode="link">
      <dt>
        <a>
          <xsl:text disable-output-escaping="yes">
            <xsl:attribute name="href">
              <xsl:value-of select="property[@name='path']/string"/>
            </xsl:attribute>
          </xsl:text>
          <xsl:attribute name="target">
            _blank
          </xsl:attribute>
        </a>
      </dt>
    </xsl:template>
  </xsl:text>
</xsl:template>
APPENDIX D. THE STYLESHEETS
D.12 Link

<!--This is the template snippet responsible to create the correct rendering of a cmlink.-->
<xsl:template match="link">
  <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called -->
  <xsl:text disable-output-escaping="yes">
    &lt;!--template rule matching a cmlink--&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
    <xsl:template match="/webobject[@oid='</xsl:text>
  <xsl:text disable-output-escaping="yes">
    <xsl:value-of select="@parent_id"/>
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    '/]//webobject[@oid='</xsl:text>
  <xsl:text disable-output-escaping="yes">
    '</xsl:text>
  <xsl:text disable-output-escaping="yes">
    '/webobject[@oid='</xsl:text>
  <xsl:text disable-output-escaping="yes">
    '</xsl:text>
  <xsl:text disable-output-escaping="yes">
    &quot;'/&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
  <a>
    <!--the url -->
    &lt;xsl:attribute name="href"&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
    &lt;xsl:value-of select="property[@name='target']/webobject/property[@name='reference']/string" /></xsl:text>
  <xsl:text disable-output-escaping="yes">
    &lt;/xsl:attribute&gt;</xsl:text>
  <xsl:variable name="resource_id">
    &lt;xsl:value-of select="child::*[@attribute:oid]"&gt;</xsl:value-of>
  </xsl:variable>
  <xsl:text disable-output-escaping="yes">
    &lt;xsl:apply-templates select="@oid"&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
    &lt;xsl:value-of select="@oid"&gt;</xsl:value-of>
  </xsl:text>
  <xsl:text disable-output-escaping="yes">
    '/property[@name='resource']/webobject[@oid='</xsl:text>
  <xsl:text disable-output-escaping="yes">
    '</xsl:text>
  <xsl:text disable-output-escaping="yes">
    \$resource_id'/&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
  &lt;/xsl:apply-templates&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
  &lt;/xsl:variable&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">
  &lt;/xsl:template&gt;</xsl:text>
  &lt;/xsl:text&gt;
  &lt;/xsl:template&gt;
D.13 Navigation

<!--This is the template snippet responsible to create the correct rendering of a cmnavigation. -->
<xsl:template match="navigation">
  <!--template rule matching a cmnavigation-->
  <xsl:text disable-output-escaping="yes">
    <!--template rule matching a cmnavigation-->
    <xsl:template match="webobject[@type='cmnavigation']">
      <table align="center" border="0" cellpadding="0" cellspacing="0" width="220">
        <tr>
          <td height="1" width="1">
            <img border="0" height="1" width="1">
            <xsl:attribute name="src">
              <xsl:value-of select="$picturepath"/>
              spacer.gif
            </xsl:attribute>
          </td>
          <td height="1" width="5">
            <img border="0" height="1" width="5">
            <xsl:attribute name="src">
              <xsl:value-of select="$picturepath"/>
              spacer.gif
            </xsl:attribute>
          </td>
          <td height="1" width="208">
            <img border="0" height="1" width="208">
            <xsl:attribute name="src">
              <xsl:value-of select="$picturepath"/>
              spacer.gif
            </xsl:attribute>
          </td>
        </tr>
      </table>
      <img border="0" height="1" width="1">
      <xsl:attribute name="src">
        <xsl:value-of select="$picturepath"/>
        spacer.gif
      </xsl:attribute>
    </xsl:template>
  </xsl:template>
</xsl:template>
D.13. NAVIGATION

</td>
<td height="1" width="5">
   <img border="0" height="1" width="5">
   <xsl:attribute name="src">
     \$picturepath\spacer.gif
   </xsl:attribute>
</img>
</td>
<td height="1" width="1">
   <img border="0" height="1" width="1">
   <xsl:attribute name="src">
     \$picturepath\spacer.gif
   </xsl:attribute>
</img>
</td>
</tr>
<tr>
   <td align="left" class="frame" colspan="5">
      <h3 class="frame">
         <xsl:value-of select="property[@name='title']"></h3>
      </td>
   </tr>
</table>
<table>
  <tr>
    <td class="frame" colspan="5" height="1" border="0" width="1">
      &lt;xsl:attribute name="&quot;src&quot;"&gt;
      &lt;xsl:value-of select="&quot;\$picturepath&quot;"&gt;
      spacer.gif&lt;/xsl:attribute&gt;
      &lt;/xsl:text&gt;
    </td>
  </tr>
</table>
D.14 Default Container

<!--This is the template snippet responsible to create the correct rendering of a default container. -->
<xsl:template match="defaultContainer">
  <!--First we create the Template-header with the correct oid, name and parent_id to make sure only the correct stylesheets are called -->
  <xsl:text disable-output-escaping="yes">&lt;!--template rule matching a cmcollection--&gt;
  </xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;xsl:template match="//webobject[@oid='</xsl:text>
  <xsl:value-of select="@parent_id"/>
  '<@oid']/" >
  </xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;xsl:text>
  </xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;xsl:apply-templates select="layout">
  </xsl:text>
  <xsl:with-param name="parent_id">
    <xsl:value-of select="@oid"/>
  </xsl:with-param>
  <xsl:with-param name="backgroundColor">
    <xsl:value-of select="@backgroundColor"/>
  </xsl:with-param>
  <xsl:apply-templates>
    <xsl:text disable-output-escaping="yes">
      &lt;/xsl:template>
    </xsl:text>
  </xsl:apply-templates>
  <xsl:text disable-output-escaping="yes">
    &lt;/xsl:template>
  </xsl:text>
</xsl:template>
D.15 Default Component

<!--This is the template snippet responsible to create the correct rendering of a default component. -->
<xsl:template match="defaultComponent">
    <xsl:text disable-output-escaping="yes">
        <
        <!--template rule matching a default component-->
        &lt;!--template rule matching a default component-->
        </xsl:text>
    </xsl:template>
Appendix E

Acknowledgements

I owe a special thank to some people for helping me writing this diploma thesis.

Prof. Moira Norrie For giving me the opportunity to do my diploma thesis in her group.

Michael Grossniklaus For his great support as my supervisor during this diploma thesis. His door was always open and he provided me with a lot of input for this thesis. A special thank for reading and correcting this report.

Sandra Kessler Just for being with me.

My family For their support and their interest.

STV Luzern For being a great handball team and a good subject for a test website.

Alan Ettlin For drinking coffee with me every day during this diploma thesis.
Bibliography


