xAPS Web Developer Manual

Version 1.8.4

Table of Contents

1. Document Introduction

1.1 Document Purpose

The purpose of this document is to give a tutorial on how to further develop xAPS Web.

1.2 Document Audience

This document is written for Java developers (employees of Ping Communication AS) who is managing the further development and improvement of xAPS Web. Could also be of interest for in-house or external web developers and/or designers.

1.3 Document History

Version	Editor	Date	Changes
1.8.4	Jarl André Hübenthal	30-06-11	Initial doc release

1.4 Acronyms and Abbreviations

None at the moment

1.5 References

Document
[1] xAPS Explained

2. Introduction

This document is created to make life easier for you as a developer and a maintainer of xAPS Web. You might ask yourself why there is a document for a simple web application, but there is a lot of ground to cover to make yourself familiar with the outs and ins of the whole process, from server side calls down to the xAPS DBI library to high level use of JavaScript to leverage the power of Ajax (with the use of jQuery). In the middle of this you even got the FreeMarker templating engine, and it can for sure become a little confusing where to look when an error occurs. But, when you have familiarized yourself with all this tooling, the separation of code and concern will be a delightful breeze in your every day battle to keep the bugz down.

It is an important guide line that one should always check the logs for xAPS Web before going into debugging and stepping. This is because in some cases problems is logged at warning level and not thrown as exceptions. A good tip is to turn on the console log for xAPS Web while testing, so that the developer can instantly see whats going on in the console within Eclipse.

3. Servlet engine

xAPS Web is managed by a normal Java Servlet, for ordinary page loads, and a Spring Servlet Dispatcher for asynchronous requests from JavaScript. Usually when creating new pages that require asynchronous request handling, annotations is added to the normal Java Servlet page, so that Spring can automatically inject URL mappings for the given page.

Creating a new page is done in four steps where the last two steps is optional (it might not be necessary to display it in the menu or to map methods to Spring):

1. Create a new class[1] that extends AbstractWebPage[2].
    - This class must implement the method process();
    - But it can also implement other methods depending upon your need.
2. Add a new entry to the Page enum[3].
    - Choose a well defined name for the enum. It can be later accessed directly in the template, so a good name makes the templates easier to read.
    - See the constructor API for help, or use the existing enums as reference.
3. Update the method createMenuItems() or getToolsMenu() in the MenuServlet[4].
    - If you want it to be visible in either the tools menu (upper right) or the main menu (middle left) you need to update the appropriate method.
    - OPTIONAL
4. Add @Controller annotation and @RequestMapping annoation to the page class
    - Also map any methods you want to be accessible from Spring with the @RequestMapping annoation. Also add the @ResponsBody annotation to the method declaration so that it will be delivered outside of the view templating system. From this method you can return any type of objects as long as it is serializable. It will be automatically converted to JSON. This can be customized by updating the file app.xml in the WEB-INF directory.
    - OPTIONAL

1 In the “com.owera.xaps.web.app.page” package
2 A contract for all Servlet pages in xAPS Web.
3 In the “com.owera.xaps.web” package
4 In the “com.owera.xaps.web.app.menu” package

4. FreeMarker engine

4.1 Introduction

The Servlet uses the FreeMarker templating engine for view processing. The view is passed a map containing keys and objects that is then accessed within the template in rendering phase. All this and more about the FreeMarker Templating Engine can be read about on the FreeMarker web page.

The general idea behind using a templating technology like FreeMarker is to split view and business logic, but at the same time not going into the JSP technology that will inevitably grant the developers too much freedom and in turn will make the application more difficult to maintain. It is important to understand that the FreeMarker templates only does the view rendering and cannot contain custom Java code or any custom code at all except for the FreeMarker language. That said you can create macros and functions, and this is relied upon heavily in xAPS Web.

In this section I will cover all the macros that has been predefined for all pages in xAPS Web, to make it easier to create forms and selects and so on. It can be a bit daunting at first sight, but when you are getting used to it it makes everything more readable.

I will also cover the parameter tables and how they they are maintained and created.

4.2 Parameter tables

4.2.1 Introduction

The list of table elements for the parameter tables in the templates[5] is generated using the TableElementMaker factory. This class contains a set of static helper methods for creating specific types of table elements. See the documentation for the TableElementMaker for more information about the API.

The parameter tables are all based on a set of classes under the com.owera.xaps.web.app.table package. The most interesting class is the one that represents the actual table row, TableElement. This class consists of basically only four or five variables at any one time:

1. The name.
    - will contain either a complete or a partial unit type parameter name.
2. The short name
    - We want to display the last part of the parameter name, so we calculate this server side.
3. The left margin.
    - or tab as it is called in-code. Must be calculated on server side due to complexity in recursive table element generation.
4. The unit type parameter
    - can contain a NULL reference, because some table rows are only parents
5. The specific parameter
    - Can be either a Unit, Profile, Group or Job parameter. No sub classing has been done for this case, so TableElement represents logically five types: Unit Type Parameter, Unit Parameter, Profile Parameter, Group Parameter and Job Parameter. The template checks for the presence of what it requires and uses. By supplying a list of profile table elements to a group parameter table, the template will not produce an expected result.

Some other variables are also set for each table element, but the above variables represents the most interesting ones.

This includes the most basic introduction for understanding how the parameter tables are generated.

A note to make everything absolutely nailed down: The parameter tables are not generated based on the toString() of the table elements. They are each iterated over in the template and bean attributes[6] are accessed at different stages in this loop.

---

5 Some templates, like for example /WebContent/WEB-INF/templates/unit/details.ftl or /WebContent/WEB-INF/templates/profile/details.ftl contains at the bottom of the template a good chunk of code for presenting the parameter tables. This chunk of code depends on a list of TableElements.

6 A term for accessing getters and setters by only specifying the attribute name, for example instead of calling person.getName() you instead call person.name. The attribute name can actually be something else, because the freemarker templates only checks for the presence of a person.getSomething when asked for person.something, reflectively.

4.2.2 An example

The Unit Type Configuration page contains a dead simple parameter table. It is the easiest template to understand. By using this as the first entry point for understanding the parameter tables, reading more advanced tables like the group parameter table will not be so difficult.

```
<table class="parameter" id="results">
	<tr>
		<th align="left">
			<a href='#' onclick='collapse();'><img src='images/minus.gif' (...) /></a>Name
		</th>
		<th>Flags</th>
		<th>
			<span title='Click here to check all' class='tiptip' onclick="check('delete')">Delete</span>
		</th>
	</tr>
	<#list params as param>
	<#if param.unittypeParameter??>
	<tr id="${param.name}">
		<td class="unittype">
			<span style="margin-left:${param.tab}px">
			<a onclick="(...)" href="javascript:void();">${param.shortName}</a>
			<img style="(...)" src="images/cantedit.gif" alt="locked" id="lock::${param.name}" title="(...)" />
			</span>
		</td>
		<td class="flagcell">
			<input name="update::${param.name}" type="text" value="${param.unittypeParameter.flag.flag}" (...) />
			<input name="update::${param.name}::Cache" type="hidden" value="${param.unittypeParameter.flag.flag}" />
		</td>
		<td align="center">
			<input name="delete::${param.name}" type="checkbox" onclick="toggle('update::${param.name}',this);" />
		</td>
	</tr>
	<#else>
	<tr id="${param.name}">
		<td>
			<span style="margin-left:${param.tab-4}px">
				<img src="images/minus.gif" onclick="TABLETREE.collapse('${param.name}')" />
				${param.shortName}
			</span>
		</td>
		<td> </td><td> </td>
	</tr>
	</#if>
	</#list>
	<tr><td colspan="3"> </td></tr>
	<tr>
		<td colspan="3" align="right">

			<input name="formsubmit" value="Update parameters" type="submit" />
		</td>
	</tr>
</table>
```

Some parts are left out for readability. The sections that is left out is replaced with “(...)”.

As you can see there is an IF test on param.unittypeParameter. This check is to test if this is a parent or if its a specific parameter. It is displayed differently in these two cases. You can easily observe that there have been added some custom pixel pushing in the else part of the if where you see style="margin-left:${param.tab-4}px". Note: the adjusted left margin could have been calculated on the server side.

Another thing to mention is how to parameters is bound to the server side. As you can see there is appended some special prefixes to the inputs, “update::” and “delete::”. If the checkbox “delete::” is not checked, it will not take part in the POST. Same goes with “create::” on other types of parameter tables, like unit or profile parameters, where parameters can be unconfigured.

4.2.3 Summary

The parameter tables have a potential for improvement. But they are fairly easy to modify. Since they are not changed very often, this document seeks to make it possible to pick up where I left.

4.3 Macro API

In this section we take a deeper look into the macros that form the library and get some dirt on hour hands by looking at some elaborate examples and verbose lists of overridable attributes. Grab a coffee.

4.3.1 Introduction

The xAPS Web FreeMarker Macro library is a time saver while creating and maintaining page templates. The key for understanding why is to dig into what makes a page template and what types of HTML code that repeats automatically. Today the usual boiler plate is SELECT and FORM. By creating and using reusable macros for forms and selects, we enable a common functionality and a faster prototype phase.

4.3.2 An example

By using the macro library the templates will become smaller, and they will essentially only contain page layout and the visual aspect of the page elements, and not too much server logic and data binding[7]. This is how it would look like without macros, using the Unit Type page as an example:

```
<html>
	<body>
		<form  autocomplete="off" action="web?page=unit-type-configuration"
method="POST"  name="form1" class="unit" accept-charset="ISO-8859-1">
			<input type="hidden" name="page" value="unit-type-configuration" />
			<select name=”unittype” id=”unittype”>
				<option value=”.”>Select Unit Type</option>
				<#list unittypes.items as unittype>
				<option value=”${unittype.name}” <#if (..check if
selected...)>selected=”selected”</#if>>${unittype.name}</option>
				</#list>
			</select>
		</form>
	</body>
</html>
```

The above can be written a lot shorter and quicker without all the boiler plate cluttering the template. Lets try to convert this to a template:

```
<html>
	<body>
		<@macros.form>
			<@macros.dropdown list=unittypes default=”Select Unit Type” />
		</@macros.form>
	</body>
</html>
```

As you can see a lot of code has been omitted and the simple reason behind this is that the

7 The SELECT macros (@macros.dropdown) is automatically creating a select element with an ID and NAME that corresponds to the Input element provided to the macro. Thus saving a great deal of time.

form macro is automatically adding all the attributes that is standard for all pages. If something needs to be changed it can be overridden. You can see this in the use of the default attribute on the dropdown macro above. The output of this template is pretty much the same as the verbose example we presented first.

4.3.3 DROPDOWN

Description

The “dropdown” macro is a helper macro for creating SELECT elements in a template. By minimizing the amount of work needed to create a select, the only work left is to create the object that extend InputSelectionModel[8] and pass it to the macro. The macro does not use nested elements so it should for the sake of readability be closed immediately.

NOTE: It expects by default that the generic type of the provided InputSelectionModel has the getName() method. If not, the template will fail tremendously. Please override callMethodForKey to “” (empty) to disable this default behavior if you per example is providing an InputSelectionModel of generic type String.

Usage skeleton

```
<@macros.dropdown list=(...) […..] />
```

Some examples

```
<@macros.dropdown list=unittypes default=”Select Unit Type” />
```

The above example will render a single select with the first option having the text “Select Unit Type”.

```
<@macros.dropdown list=unittypes size=”5” multiple=true />
```

The above example will render a multiple select with 5 visible elements. No default option.

Required attributes

- list
  - An object extending InputSelectionModel, DropDownSingleSelect or DropDownMultiSelect, or similar type of objects with the same getters and setters.
  - Use InputSelectionFactory to generate the InputSelectionModel object.

Overridable attributes

- default
  - Does not mean “default selected”[9].
  - The display value for the default OPTION.
    - If specified it will create an extra OPTION in the SELECT.
    - If not specified it is ignored by the macro.

8 This cannot be created directly, so you must use the InputSelectionFactory to create instances of it.
9 The default selected must be provided by the list object.

- defaultValue
  - Only effective if the default attribute above is specified.
  - Specifies the value attribute for the default OPTION.
  - Defaults to “.” that translates to “no value”.
  - Should not be changed without a very good reason.
- callMethodForKey
  - The value for the OPTION
  - Specifies the attribute to retrieve reflectively for each item in the dropdown
  - Defaults to “name”.
  - If empty (“”) it resolves to “toString()”.
- callMethodForDisplay
  - For the display of the OPTION
  - Defaults to callMethodForKey behavior
- not
  - A list of objects to exclude from this SELECT
  - Only used for simple String selects
  - Defaults to empty array.
- disabled
  - TRUE/FALSE. Switch for whether or not to be disabled.
  - Defaults to FALSE.
- encodeValue
  - TRUE/FALSE. In some special cases we need to HTML encode the values of the options
  - Defaults to FALSE.
- class
  - set the class attribute
- namePrefixForId
  - Some selects needs a prefix for the id attribute
  - Defaults to empty
- size
  - A number as String
  - Set how many options should be visible
  - Defaults to “1”

- multiple
  - TRUE/FALSE. Enable or disable multiple select behavior.
  - Defaults to FALSE.
- visible
  - TRUE/FALSE. Hide or display the select.
  - Defaults to TRUE.
- width
  - Defaults to “200px”
- onchange
  - Defaults to “processForm('form1')”
  - Note: It should be converted to “xAPS.submitForm()”
    - If you name your form something else than “form1” you should also override onchange to “xAPS.submitForm('yourFormName')”
  - Disables default behavior with empty content (“”).

4.3.4 FORM

Description

The “form” macro is a helper macro for creating FORM elements in a template. This is one of the types of elements that repeat very often, so by setting default values that is page sensitive[10], we can make sure that all forms behave in the same manner.

Usage skeleton

```
<@macros.form […..]>
        [….]
</@macros.form>
```

Some examples

```
<@macros.form>
        <@macros.dropdown list=unittypes default=”Please select me” />
        <input type=”submit” name=”formsubmit” value=”Click me please” />
</@macros.form>
```

The above example will render a form with the current page in the action attribute with also a hidden page field, a dropdown for unittypes with the first option having “Please select me” and a submit button with the text “Click me please”.

Required attributes

- Nothing is required for it to work out of the box

Overridable attributes

- onsubmit
  - Defaults to empty (none)
- name
  - Defaults to “form1”
- autocomplete
  - Defaults to “off”
- id
  - Defaults to empty (none)
- upload
  - Defaults to FALSE.
- class

10 The form macro will automatically, if there is a CURRENT_PAGE variable in the template, add an action attribute, (action=”web?page=${CURRENT_PAGE.id}”) and also a hidden field (<input type=”hidden” name=”page” value=”${CURRENT_PAGE.id}” />)

    - Defaults to “unit”
- method
    - Defaults to “POST”
- action
    - Can override the automatic action generation
    - Defaults to empty (will then use the automatic action generator).
- charset
    - Defaults to “ISO-8859-1”
- fieldset
    - Wrap the form in a fieldset with this title
    - Defaults to empty (none).

4.3.5 CHECKBOX

Description

The “checbox” macro is a helper macro for creating INPUT elements of type checkbox.

Usage skeleton

<@macros.checkbox list=(...) />

Some examples

<@macros.checkbox class="needsgraphreload" namePrefixForId="${type}" list=Sysaggregation onclick="" />

This is an example from /WebContent/WEB-INF/templates/unit-status/chart.ftl. It creates a group of checkboxes that appends the assigned namePrefixForId to the name of each checkbox. The example overrides onclick to empty, which is normally set to submit the form.

Required attributes

- list
  - An object extending InputSelectionModel, CheckBoxGroup, or similar type of object with the same getters and setters.

Overridable attributes

- class
  - Set the class attribute
  - Defaults to empty (none)
- callMethod
  - Define what method to call reflectively on each checkbox item
  - Defaults to empty (toString() behavior)
- suffix
  - Define what html to append after each checkbox
  - Defaults to “
”
- namePrefixForId
  - A badly designed name, should be prefixForName
  - Defaults to empty (none)
- onclick
  - Defaults to “processForm('form1');”

4.3.6 RADIO

Description

The radio macro is a helper macro for creating INPUT elements of type radio.

Usage skeleton

<@macros.radio list=(...) />

Some examples

<@macros.radio class="submitonchange" list=types onclick="" suffix=" " callMethod="" />

This is an example from /WebContent/WEB-INF/templates/report/custom/Group.ftl. It creates a group of radio buttons with a space as suffix and by calling toString() on each type. Disable normal onclick.

Required attributes

- list
  - An object extending InputSelectionModel, CheckBoxGroup, or similar type of object with the same getters and setters.

Overridable attributes

- class
  - Set the class attribute
  - Defaults to empty (none)
- callMethod
  - Define what method to call reflectively on each checkbox item
  - Defaults to empty (toString() behavior)
- suffix
  - Define what html to append after each checkbox
  - Defaults to “
”
- namePrefixForId
  - A badly designed name, should be prefixForName
  - Defaults to empty (none)
- onclick
  - Defaults to “processForm('form1');”

4.3.7 HIDDEN

Description

The “hidden” macro is a helper macro for creating INPUT elements of type hidden.

Usage skeleton

```
<@macros.hidden list=(...) />
```

Some examples

```
<@macros.hidden list=groups />
```

This is an example from /WebContent/WEB-INF/templates/group/details.ftl. It creates a hidden field based on the supplied InputSelectionModel.

Required attributes

- items
  - An object extending InputSelectionModel, or similar type of object with the same getters and setters, the same type of objects that is used in conjunction with the DROPDOWN macro.

Overridable attributes

- callMethodForSelected
  - Set the method name to call reflectively on each item
  - Defaults to “name”

4.3.8 TEXT

Description

The “text” macro is a helper macro for creating INPUT elements of type text. _Since the onchange attribute is not used, so no overridable attribute is provided for that. This can be easily added at a later point in time.

Usage skeleton

```
<@macros.text input=(...) />
```

Some examples

```
<@macros.text input=groups.input size="30" />
```

This is a dummy example. It creates a text field of size 30 by passing in the input for the groups dropdown.

Required attributes

- input
  - An object extending Input

Overridable attributes

- size
  - Set the size of the text field
  - should be number, but the macro wont crash with a string
  - Defaults to empty (none)
- onkeyup
  - sets the onkeyup attribute
- style
  - sets the style attribute

4.3.9 Summary

In short the amount of work saved by using this library should be a ringing bell for all of those that still believe that everything should be made from ground up. By focusing on creating pages that behave in the same way we can minimize the amount of bugz and the time needed for getting things up. In my experience the real work is pixel pushing and css/layout. The core page functionality should be quick to get up, easy to change and should just work without too much fuzz.

5. JavaScript engine

There is a lot of JavaScript within xAPS Web. Actually at such a degree that the application is rendered useless if JS disabled in the browser. Therefore a box will alert the user if JS disabled, so that the user can know why things are not working as they should.

The xAPS JavaScript Library is documented in the JavaScript files directly, so this document section will only cover the conceptual ideas and the architecture in general. Also the connection to the templates and how it all operates towards the Java Servlets.

5.1 JQuery TableSorter plugin

Introduction

Most pages in xAPS Web contains tables that needs client side sorting capabilities. This is made possible with jQuery Tablesorter plugin. The plugin greatly improves the usability of the various data tables in xAPS Web.

For specific documentation for this plugin, visit the plugin home page found at jquery.com.

This document only sheds light on how it is used within xAPS Web, and what things that has been changed/altered from the original script.

Basic usage

The script supports configuring what columns should be default sorted and what direction they should be sorted in. This can be specified in the class attribute as seen below, with the sortlist meta attribute. The array [1,1] tells sortlist to sort the second column with the highest value on top (descending). An array of [1,0] would mean sorting the second column ascending.

```
<table class="tablesorter {sortlist:[[1,1]]}">
	<thead>
		<tr>
			<th>Unit</th>
			<th>Count</th>
		<tr>
	</thead>
	<tbody>
		<tr>
			<td>unit 1</td>
			<td>3</td>
		<tr>
		<tr>
			<td>unit 2</td>
			<td>1</td>
		<tr>
		<tr>
			<td>unit 3</td>
			<td>5</td>
		<tr>
		<tr>
			<td>unit 4</td>
			<td>4</td>
		<tr>
	</tbody>
</table>
```

Sorting on multiple columns

You can also combine multiple columns to create complex sorting behaviors, like in the below example where we first sort on the second column descending, then the fourth column ascending and then the third column descending:

```
<table class="tablesorter {sortlist:[[1,1],[3,0],[2,1]}">
	<thead>
		<tr>
			<th>Unit</th>
			<th>Count 1</th>
			<th>Count 2</th>
			<th>Count 3</th>
		<tr>
	</thead>
	<tbody>
		<tr>
			<td>unit 1</td>
			<td>3</td>
			<td>23</td>
			<td>36</td>
		<tr>
		<tr>
			<td>unit 2</td>
			<td>1</td>
			<td>10</td>
			<td>3</td>
		<tr>
		<tr>
			<td>unit 3</td>
			<td>5</td>
			<td>3</td>
			<td>1</td>
		<tr>
		<tr>
			<td>unit 4</td>
			<td>4</td>
			<td>3</td>
			<td>2</td>
		<tr>
	</tbody>
</table>
```

Paging for browsing large tables

You need to manually configure tablesorter pager plugin for your table within a script tag below the table as shown below. It is not automatically configured for all tables, because not all tables need paging.

```
<table class="tablesorter {sortlist:[[1,1],[3,0],[2,1]}">
	<thead>
		<tr>
			<th>Unit</th>
			<th>Count 1</th>
			<th>Count 2</th>
			<th>Count 3</th>
		<tr>
	</thead>
	<tbody>
		<!-- Many rows -->
	</tbody>
</table>
<div id="pager" class="pager">
	<form>
		<img src="images/pager/first.png" class="first"/>
		<img src="images/pager/prev.png" class="prev"/>
		<input type="text" class="pagedisplay"/>
		<img src="images/pager/next.png" class="next"/>
		<img src="images/pager/last.png" class="last"/>
		<select class="pagesize">
			<option value="100">100</option>
			<option value="500">500</option>
			<option value="1000">1000</option>
			<option value="5000">5000</option>
			<option value="10000">10000</option>
		</select>
	</form>
</div>
<script>
	jQuery(document).ready(function($){
		$(".tablesorter").tablesorterPager({container: $("#pager"),size: 100});
	});
</script>
```

Configuring custom sorting values

Some table columns contain data that is not automatically possible to sort. You then have to specify a value to sort on by specifying the “tablesorter_customkey” attribute on the TD element. In the example below we specify the amount of milliseconds to sort on because elapsed time is presented in a human readable format not suitable for sorting.

```
<table class="tablesorter">
        <thead>
                <tr>
                        <th>Unit</th>
                        <th>Elapsed time</th>
                <tr>
        </thead>
        <tbody>
                <tr>
                        <td>unit 1</td>
                        <td tablesorter_customkey=”123456789”>34 days 24 hours</td>
                <tr>
        </tbody>
</table>
```

Things that has been changed in the original script

In the method getElementText(config,node), I have added a block for checking if the parent TD has a “tablesorter_customkey” attribute. If it exists it is returned as the text to be used for sorting.

```
function getElementText(config, node) {
            (….)
            if (config.textExtraction == "simple") {
               var customValue = $(node).closest("td").attr("tablesorter_customkey");
               if (customValue){
                    return customValue;
               } else if (….)
            } else {
               (….)
            }
            return text;
}
```

5.2 xAPS Module Library

For simpler management of the JavaScript contained in the various pages in xAPS Web, there has been created a JavaScript library to encapsulate all the JavaScript code used. Not all code is defined within this library, but it is a good rule to follow that each page should have its own JavaScript container.

All documentation related to how the library works and the ideas behind it, can be found in the JavaScript source files. There is not need to repeat that documentation in this document.

6. Summary

In general the documentation contained in this document is just scratching the surface. You can find more documentation by looking at javadoc and existing pages. Two pages to use as examples are the Report and Unit Dashboard/History page. They are both reasonably complex.