MMBase Editwizard Front end

Nico Klasens, Finalist IT Group

Table of Contents

Introduction

The MMBase editwizard extension provides an easy way to create customizable wizards for edit environments. In most cases the default editwizard look does not match the look of the rest of the edit environment. This document describes how the look and feel of the editwizard front end can be changed to match with the rest.

Assumptions

The front end relies heavily on xsl, css and javascript. Therefore, this document assumes you know something about these techniques. A good reference for these techniques is http://www.w3schools.com and the official w3c site (http://www.w3c.org)

Inner workings

We will elaborate a little on the inner workings of the editwizard extension before we dive into the details of the front end. The editwizard extension can be divided into 2 layers. The first layer is the server side engine which consists of java and jsp files. The engine communicates with MMBase and generates the data for the second layer. The second layer is the front end, which runs on the server and in the client browser.

The engine communicates with MMBase through the dove interface. The dove interface adds a protocol for communication with, and passing commands to, the MMBase system using an XML format. The output from the engine to the front end is an XML stream with the data to build the html pages. The engine is not only a pass-through for data. It also keeps track of stacks with opened wizards for every user and parses the wizard definition schema files.

Files and directory locations

The editwizard extension assumes a directory structure where files can be found. Two root locations are used to find the files. The most important one is the editwizard home directory. The editwizard home is the directory that contains the editwizard data library, css-stylesheets, javascript files and

jsp pages. In the standard distribution, this location is /mmapps/editwizard/. The other root location is the referrer page directory. You call the editwizard scripts (list.jsp and wizard.jsp) from a page located in another directory than the editwizard home directory. If you do, the wizards keep a reference to the calling page, the 'referrer'. When a xml or xsl file is requested, the wizard first checks whether the file can be found in the directory of the referrer page. You can also use this location to have you own javascript files or css-stylesheets.

The assumed sub locations of the editwizard home directory are:

- /data/

  Searched for wizard definition schema files when they are not found in the referrer page directory.

- /data/i18n/

  Translated prompts in the wizards. These will override the English version in the /data/xsl/ directory

- /data/xsl/

  Xsl transformation files used to generate the html pages. These are used when the xsl files are not found in the referrer page directory.

- /javascript/

  Javascript files which are used for many things on the client side.

- /jsp/

  scripts to enter the editwizard extension. Frequently used files are list.jsp and wizard.jsp.

- /media/

  Images and other media files used in the front end.

- /style/color/

  css-stylesheets which define the colors of the wizards.

- /style/layout/

  css-stylesheets which define the layout of the wizards.

Types of pages

The editwizard generates five types of pages. Three of them are used in normal operation. These are the list, wizard and searchlist pages. Most of the times, you go from a list page to a wizard page and then you add related objects with the searchlist page to the wizard. The ones left over are the debug and error (exception) pages. The debug page shows editwizard engine internal data which could be handy to use when changing the front end xsl's. Hopefully, you will never see the error page. If you look in some directories then you will see files with the same names as the types of the pages.

Changing the look

You probably already guessed what to do when you want to change the colors and images of the front-end. The colors are defined in the css-stylesheets in the /style/colors/. This is the only place where colors are defined. If you want to know where the class definitions are used then you should consult the comments in /style/layout/wizard.css. The images are in the /media/ directory so that is not hard to change too. If you want to change the editwizard hardcoded prompts (eg the save and

cancel link) then you should change the /data/xsl/prompts.xsl (and the i18n ones) to accomplish this.

The above are the simple things to change your wizards. You can customize much more. One of the powerful things in the editwizard extension is that you can use the referrer page directory to override the default xsl's. One big pro is that you can easily change to a newer version of the editwizard extension without re-applying your own changes. The xsl's have empty templates (eg. extrameta, extrastyle, extrajavascript) where you can add extra functionality for your wizards and variables to turn things on and off. One of these variables is the SEARCH_LIST_TYPE which switches the searchlist from a popup window to an iframe.

Example 1: logo in the header

This example shows how you can add a logo to the header of the wizards. The first thing you have to do is create a file base.xsl in your referrer directory. The new base.xsl will import the one from the editwizard home directory. The ew: prefix in the import points to the editwizard home directory. The new xsl overrides the headcontent template from the editwizard base.xsl to add an extra column for the logo.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:node="org.mmbase.bridge.util.xml.NodeFunction"
  xmlns:date="org.mmbase.bridge.util.xml.DateFormat"
  extension-element-prefixes="node date">

   <!-- Import original stylesheet -->
   <xsl:import href="ew:xsl/base.xsl"/>

  <xsl:template name="headcontent" >
    <table class="head">
      <tr class="headtitle">
        <td rowspan="2"><img src="{$referrerdir}my_logo.jpg"/></td>
        <xsl:call-template name="title" />
      </tr>
      <tr class="headsubtitle">
        <xsl:call-template name="subtitle" />
      </tr>
    </table>
  </xsl:template>

</xsl:stylesheet>
```

Example 2: new fieldtype

Adding a logo is not so hard to do, but what if you have your own fieldtype (wizard definition schema attribute ftype)? Maybe, you want your fieldtype represented as a checkbox, radio button or dropdown box. This involves graphical representation, client-side initialisation and validation.

The first thing we have to do for this is overriding the ftype-unknown template of the wizard.xsl (See the first example). This template calls templates for unknown ftypes to generate the graphical representation.

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
  xmlns:node="org.mmbase.bridge.util.xml.NodeFunction"
  xmlns:date="org.mmbase.bridge.util.xml.DateFormat"
  extension-element-prefixes="node date">

   <!-- Import original stylesheet -->
   <xsl:import href="ew:xsl/wizard.xsl"/>

  <xsl:template name="ftype-unknown">
    <xsl:choose>
      <xsl:when test="@ftype=&apos;my_ftype&apos;">
              <xsl:call-template name="ftype-my-ftype"/>
      </xsl:when>
      <xsl:otherwise>
              <xsl:call-template name="ftype-other"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="ftype-my-ftype">
    <input type="checkbox" name="{@fieldname}" value="{value}" id="{@fieldname}
      <xsl:apply-templates select="@*" />
    </input>
  </xsl:template>

  <xsl:template name="extrajavascript">
    <script language="javascript" src="{$referrerdir}override.js"></script>
  </xsl:template>

</xsl:stylesheet>
```

Now the editwizard will generate a checkbox for all fields with ftyp=”my_ftype”. We will use an <referrer page directory>/override.js to initialise and validate the checkbox. The validator will attach itself to the input fields when it is required to validate it. The validation is now only at the client-side. The editwizard will do a server-side validation for the existing fieldtypes too.

```
function initializeElement(elem, dttype, ftype) {
    if (ftype == "my_ftype") {
        elem.checked = true;
    }
}

//validator methods
function requiresUnknown(el, form) {
    return (el.getAttribute("ftype") == "my_ftype");
}

function validateUnknown(el, form, v) {
    if (el.getAttribute("ftype") == "my_ftype" && el.checked) {
        return "Checkbox is still checked";
    }
    return "";
}
```