

Component Framework

Date 2006-10-13

Edition \$Id: components.xml,v 1.0 2006/10/03 08:17:09 henk Exp \$

Authorgroup Nico Klasens

Henk Hangyi

Revhistory ▶ [0.1] [2006-10-13] [HH] [Created]

Abstract This document describes the MMBase component framework.

Legalnotice This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

The license (Mozilla version 1.0) can be read at the MMBase site. See <http://www.mmbase.org/license>

Productnumber Component Framework from MMBase 1.9.

1 Introduction

The component framework adds component based developing to MMBase. Components can be accessed from jsp-pages directly or be used in a portlet engine / portal service. For use in jsp-pages MMBase offers tags in the MMBase taglib which put the components into action and render their content into your page. When using a portlet engine and portal service, like the CMSContainer, this engine takes care of analyzing the client request, make the selected portlets execute and render their content and return the resulting page to the client.

2 Hello World!

Lets start with a simple example. The following configuration file for component "core" defines one block "components". This block has one renderer that can be used in the html <body> tag.

Note The configuration files for the mmbase core are stored in the \mmbase\config\ directory. This file can therefore be found in /mmbase/config/components/core.xml.
TODO: the files for the MMBase core should be moved to /mmbase/core/

```
<?xml version="1.0" encoding="UTF-8"?>
<component
  name="core"
  defaultblock="components"
  xmlns="http://www.mmbase.org/xmlns/component"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mmbase.org/xmlns/component
    http://www.mmbase.org/xmlns/component.xsd">
  <description xml:lang="en">MMBase core components</description>
  <block name="components" mimetype="text/html">
```

```
<body jsp="/mmbase/admin/components.jsp" />
</block>
</component>
```

The jsp-include "/mmbase/admin/components.jsp" could look like:

```
<?xml version="1.0"?>
<div xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:mm="http://www.mmbase.org/mmbase-taglib-2.0">
  <jsp:output doctype-root-element="html" doctype-public="-//W3C//DTD XHTML 1.0 Strict/"
  <mm:content type="text/html" language="en" expires="0">
    <h1>Components admin page</h1>
    Hello World!
  </mm:content>
</div>
```

To render this, a jsp-page has to contain the following tag:

```
<mm:component name="core" block="components" render="body" />
```

▼ 3 Configuration of a component

The core of a component is the component.xml. It specifies the blocks in the component and the renders within each block. The following example provides the ecards.xml for an ecard component.

```
<?xml version="1.0" encoding="UTF-8"?>
<component
  name="ecards"
  defaultblock="home"
  xmlns="http://www.mmbase.org/xmlns/component"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mmbase.org/xmlns/component
                      http://www.mmbase.org/xmlns/component.xsd">
  <description>Ecards component</description>
  <block name="home" mimetype="text/html">
    <process jsp="ecards_init.jsp" />
    <head jsp="ecards_head.jsp" />
    <body jsp="ecards.jsp" />
  </block>
  <block name="select" mimetype="text/html">
    <head jsp="ecards_head.jsp" />
    <body jsp="ecards_selectecard.jsp" />
  </block>
  <block name="done" mimetype="text/html">
    <process class="org.mmbase.ecards.Send" />
    <head jsp="ecards_head.jsp" />
    <body jsp="ecards_done.jsp" />
  </block>
```

```
</component>
```

In the above example some renderers are jsp-includes, other are java classes. The information available to the jsp files are the request parameters, session attributes and the tags in which the `<mm:content />` tag is contained. For the java classes the information is available from `HttpServletRequest` request and `HttpServletResponse` response. Btw. by using `request.getSession()` the session in which the block is rendered can be accessed.

▼ 4 The `<mm:component />` tag

This section provides an overview over the parameters and functionality of the `<mm:content />` tag.

▼ 4.1 defaults

In the "Hello World" example the following tag was used:

```
<mm:component name="core" block="components" renderer="body" />
```

The default block can be specified by using the `defaultblock` attribute of the `<component />` tag. In the "Hello World" example it is `defaultblock="components"`. If no `defaultblock` is specified the first block is considered to be the default block. The default renderer is `body`. By using the defaults the tag could be rewritten to:

```
<mm:component name="core" />
```

When renderer for a block is not specified, rendering the block will return nothing.

▼ 4.2 render

The renderers that are supported in the present implementation are: `head`, `body`, and `process`.

The `process` renderer of block is called implicitly, if the `head` or the `body` of that block is called. The `process` renderer of block will (1) only be executed once per calling page and (2) only be executed after the first changes to this block.

The renderer for `body` with `mimetype="text/html"` should render a `div` with `class="TODO look at portlet css specification"`.

▼ 4.3 mimetype

The `mimetype` can be used to indicate that the component is rendering special file types like images, attachments, etc. In the example above we could have used `mimetype="application/xhtml+xml"`. Where Firefox interpretes this filetype correctly and will check the validity of the page, the present versions of Internet Explorer will prompt the user to download this page. Because of this IE bug it is better to use `"text/html"`.

▼ 4.4 position of includes

Here the jsp include `"/mmbase/admin/components.jspx"` is positioned absolute to the root of the webapplication. By setting the request parameter `doMakeRelative` to `true`, the jsp include can be positioned relative to the jsp page that contains the `<mm:component />` tag.

▼ 4.5 parameters

When the `<mm:component />` tag is contained in another tag all the information from the containing tag is accessible to the `<mm:component />` tag. For instance in the situation `<mm:cloud jspvar="cloud"><mm:component name="core" /></mm:cloud>` the `components.jsp` could contain `<%= cloud.getUser().getIdentifier() %>`.

When it is necessary to include extra parameters this can be done by using the `<mm:param />` tag.

Some examples:

```
<mm:component name="my_game"><mm:param name="level" value="novice" /></mm:component>
```

```
<!-- this piece uses the poll component to show two polls -->
<mm:node number="first_poll"><mm:component name="poll"><mm:node>
<mm:node number="second_poll"><mm:component name="poll"><mm:node>
```

▼ 5 Framework and changing the behavior of the `<mm:url />` and `<mm:include />` tags

An MMBase framework provides the context in which components are rendered. The present functionality of the framework is that it changes the behavior of the `<mm:url />` and the `<mm:include />` tag. The next section shows how frameworks are implemented. The "Hello Again!" example gives an example of using a framework.

▼ 5.1 Framework

A framework is an implementation of `org.mmbase.framework.Framework`. By implementing the method `Framework.getUrl()` the behaviour of `<mm:url />` and `<mm:include />` can be changed. This can be used to change the layout and the includes used in the page, based on the parameters passed to that page.

The framework that will be used for rendering the components an MMBase instance is specified in `mmbaseroot.xml`, by using the parameter

```
<property name="framework"> ... </property>
```

If no framework is specified in `mmbaseroot.xml` the `org.mmbase.framework.BasicFramework` will be used.

▼ 5.2 Hello again! and goodbye to `<mm:treeinclude />` and `<mm:treefile />`

The use of the Framework functionality is shown by the following "Hello again!" example.

```
<?xml version="1.0" encoding="UTF-8"?>
<component
  xmlns="http://www.mmbase.org/xmlns/component"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mmbase.org/xmlns/component http://www.mmbase.org/xml
  name="helloworld">
  <description>Hello Again</description>
  <block name="home" mimetype="text/html">
    <head jsp="hello_head.jsp" />
  </block>
</component>
```

The jsp-include hello_head looks like:

```
<%@page language="java" contentType="text/html; charset=utf-8" session="false"%>
<%@taglib uri="http://www.mmbase.org/mmbase-taglib-1.0" prefix="mm"%>
<mm:content type="text/html" language="en">
<mm:cloud>
  <title>Hello world</title>
  <link rel="stylesheet" type="text/css" href="<mm:url page="css/hello.css" component=
</mm:cloud>
</mm:content>
```

The jsp-page that uses this component looks like:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml11-strict.dtd">
<%@page import="org.mmbase.framework.*"%>
<%@page import="org.mmbase.util.functions.*"%>
<%@page language="java" contentType="text/html; charset=utf-8" session="false"%>
<%@taglib uri="http://www.mmbase.org/mmbase-taglib-1.0" prefix="mm"%>
<mm:content type="text/html" language="en">
<mm:cloud>
<mm:import externid="component" />
<head>
  <mm:component name="hello" block="home" render="head" /> ? parameters are case-insen
</head>
<body>
  <h1>Hello again!</h1>
</body>
</mm:cloud>
</mm:content>
```

The call to `<mm:url page="css/hello.css" component="hello" />` is picked up by the `getUrl()` method of the framework which is specified for this MMBase instance.

This method

```
getUrl(String page, String component, Cloud cloud, PageContext pageContext, List param
```

could for instance call

```
UrlResolver.findUrl(component + "/" + page, cloud, pageContext, params)
```

In the `findUrl()` method the `params` can be used to select different `css-es` for different portals. The code which is used for this looks something like:

```
Node portalNode = cloud.getNode((String)params.get("portal"));
String finalpage = findUrl(page, portalNode, mapNode);
if (finalpage != null) {
  return File.separator + finalpage;
}
```

The example framework presented here thus provides the functionality to use one set of templates, but

have subsites with different layouts and subsite-specific includes.

Note If you are familiar with `<mm:treeinclude />` and `<mm:treefile />` you will probably already have recognized that this framework replaces the functionality of both these tags.

▼ 6 Action flow within the portlet engine / portal service

To give an idea of how a portlet engine / portal service works this sections gives an overview of the flow of actions that take place when a client calls an url:

1. Client calls url
2. Tomcat (or other application server) routes url to web application of the portal
3. A servlet inside the portal web application receives the url
4. Portal servlet will analyze the request
 - global navigation path to a page
 - local navigation for the portlet (indicates which portlet is active in this call)
 - window state for each portlet
 - portlet mode for each portlet
 - render parameters for each portlet
 - action parameters from query string
5. Start of action phase
 - Portal servlet resolves portlet instance which user wants to interact with
 - Portal servlet creates `ActionRequest` and `ActionResponse` objects
 - Portal servlet dispatches `ActionRequest` to the servlet which hosts the portlet instance for the action phase and sends the action parameters
 - Portlet instance processes the action parameters maybe with the help of its own presentation framework (struts, jsf, jsp, tapestry, wicket, whatever)
 - Portlet instance can change things in the request, session, preferences, external system, database, etc.
 - Portlet instance modifies `ActionResponse` to tell the Portal servlet what should happen next. Redirect to client or render phase.
 - Portal Servlet receives `ActionResponse` and acts on it
6. Start of render phase
 - Portal servlet resolves page object from global navigation.
 - Portal servlet retrieves all portlet instances on the page. For each portlet instance
 - Portal servlet creates `RenderRequest` and `RenderResponse` objects
 - Portal servlet dispatches `RenderRequest` to the servlet which hosts the portlet instance.
 - Portlet instances reads portlet mode and window state.
 - Portlet instances calls his own presentation framework (struts, jsf, jsp, tapestry, wicket, whatever).
 - Portlet instances writes markup fragment to `RenderResponse`.
 - Portal servlet retrieves page template
 - Portal servlet decorates portlet `RenderResponse` outputs with window and portlet mode buttons and inserts it in the page template.
 - Final result is written to client response

- Portal servlet returns response to the client.

In the above flow no separation is made between portal service and the portlet engine (eg pluto). The portlet engine provides the runtime environment for the portlet instances. The portal service does all page related stuff.

▼ 7 The CMSContainer: portlets and components

The CMSContainer is one of the MMBase contributions. For documentation of the CMSContainer see the reference list at the end of this document. This section addresses the relation between portlets and components.

▼ 7.1 Using components as portlets

A wrapper class will be developed in the CMSContainer, which makes it possible to use any MMBase component in the CMSContainer. This means that for the components the CMSContainer will use the components.xml from the component instead of the project.xml that is used for the portlets in the CMSContainer.

▼ 7.2 Using portlets as components

The other way around the situation is more difficult. When a portlet depends on the objectmodel of the CMSContainer it is not possible to rewrite it to an MMBase component, that can be reused outside the CMSContainer. Only portlets that do not depend explicitly on the objectmodel of the CMSContainer can be rewritten into components. The present version of the CMSContainer in the MMBase CVS does not contain examples of such portlets. But don't get worried: portlets like calender, playlist, etc. which are now being developed will be added as components later.

▼ 8 Some notes on the location of files

It is handy to use the same structure to store the files of a component within an application or contribution. When it comes to building, the exact location is of minor importance because the build process can reshuffle directories to get them into the right location in the build. Below follows an overview of how files are structured at the moment.

▼ 8.1 Applications and contributions

In the 1.8 MMBase applications and contributions files are stored in the following directories:

- MyApplication
 - config
 - ◆ applications
 - MyApplication.xml
 - MyApplication
 - builders
 - ◆ builders
 - ◆ functions
 - ◆ log

- ◆ modules
- ◆ security
- ◆ thememanager
- documentation
- packaging: what to do this?
- templates
- src
- META-INF
- WEB-INF
- readme.txt
- build.xml

▼ 8.2 The CMSContainer

The CMSContainer contains several portlets, which all can be viewed as separate applications. These portlets can be found in /contributions/CMSContainer/cmssc. For the CMSC the Maven preferred way of storing files is used. The files of these portlets are stored in the following directories:

- config
- resources: resource bundles and property files
- src
 - java
 - tld
 - webapp
- project.xml

▼ 8.3 Didactor

In Didactor the components are structured as follows

- mycomponent
 - config
 - ◆ applications
 - MyComponent
 - MyComponent.xml
 - ◆ components
 - mycomponent.xml
 - ◆ translations
 - java
 - lib
 - templates
 - webinf

▼ 8.4 Component Framework (MMBase 1.9)

For the file structure of components the following principle will be used.

- MyContribution
 - components
 - ◆ mycomponent1
 - config
 - mycomponent.xml
 - model.xml
 - builders
 - data
 - functions
 - log
 - modules
 - security
 - documentation
 - editwizards
 - templates
 - src
 - resources
 - tld / META-INF
 - WEB-INF
 - project.xml
 - readme.txt
 - ◆ mycomponent2
 - ◆ mycomponent3
 - documentation
 - build.xml
 - readme.txt

Note In the new structure the applications directory is replaced by the model.xml file in the config directory of the component and a data directory containing the default data for the component, the "apps1" xml dump.

Note The editwizards are stored in a separate directory. In this way the build process itself can move the editwizards to the directory where the target application expects them.

Note The lib directory is not necessary because the needed jar files are specified in the project.xml and will be downloaded during the build.

▼ 9 References

The following documents can be used for further reading.

- MMBase Component Framework projectpage <http://www.mmbase.org/mmcf>
- Presentation on JSR168 by Nico Klasens <http://www.mmbase.org/mmbase/attachments/50434/JSR168.zip>

- An MMBase Component Framework by Johannes Verelst
http://www.mmbase.org/mmbase/attachments/50347/MMBase_framework_-_tech.doc
- Design CMS Container by Nico Klasens http://cmssc.finalist.com/Design_CMSC.pdf or
http://cvs.mmbase.org/viewcvs/*checkout*/speeltuun/applications/cmssc/Design_CMSC.pdf
- What Is a Portlet <http://www.onjava.com/lpt/a/6208>