

# WeWebU OpenECM-Framework 3.0.0.0

## Developer Manual

Copyright

WeWebU Software AG  
Hauptstr. 14  
91074 Herzogenaurach  
Germany

Phone: +49 (9132) 83660 - 0

E-Mail: [contact@wewebu.de](mailto:contact@wewebu.de)

<http://www.wewebu-software.com>

**Version 1.0**

**Status: Approved**

**Author: WeWebU Software AG**



## Typographical Conventions Used in This Manual

Convention	Example
Standard	Standard Text
<b>Important terms</b>	<b>OpenECM-Framework</b>
<b>Classes</b>	<b>OwApplicationContext</b>
Methods	<code>getName()</code>
Source Code	<code>public void myMethod(){} // simple source code comment</code>
Comments in Source Code	
<b>Java Package</b>	<b><code>com.wewebu.server.app</code></b>
<b>Plugins</b>	<b><code>com.wewebu.ow.server.plugin.owdemo.owmain</code></b>
<b>Path</b>	<b><code>com/wewebu/ow/server/...</code></b>
XML Attribute	<code>&lt;node&gt;</code>
XML value	<code>value</code>
<b>Internal link to other chapters in the document</b>	<b>Introduction</b>
External link to other documents	INST, chapter "Installing the Configurator"
<i>Product-specific terms</i>	<i>Add Document</i>

### Formatting legend for brackets in Source Code, Paths, File Names, etc.:

Ellipsis (...): Parameter the user must supply

Between brackets [...]: Optional items

Between braces {...}, choices separated by pipe |, example: {even|odd}: Set of choices from which the user must choose only one

### The following placeholders are used in this document:

- (OpenWorkdesk) - root directory of OWD deployment.
- (Workplace) - root directory of IBM FileNet P8Workplace deployment.
- (Websphere) - installation root of Websphere application server.
- (Tomcat) - installation root of Tomcat application server.
- (JBoss) - installation root of JBoss application server.
- (ContentManager) - installation root of IBM Content Manager.





## Table of Contents

<b>1</b>	<b>Introduction</b> .....	
1.1	INTENDED AUDIENCE .....	
1.2	ABOUT THIS DOCUMENT .....	
<b>2</b>	<b>Overview Framework</b> .....	<b>6</b>
2.1	ARCHITECTURE .....	6
2.2	JAVA PACKAGES.....	8
<b>3</b>	<b>First Installation</b> .....	<b>10</b>
3.1	DELIVERY AND DEPLOYMENT STRUCTURE .....	10
3.1.1	Overview of Directory Structure.....	10
3.1.2	Developer Toolkit.....	11
3.2	SYSTEM REQUIREMENTS .....	12
3.3	INSTALLATION AND TEST .....	12
<b>4</b>	<b>Customizing Layout and Designs</b> .....	<b>14</b>
4.1	CREATING A NEW DESIGN .....	14
4.2	TESTING DESIGNS .....	15
4.3	CUSTOMIZING DESIGNS .....	15
4.3.1	Overview .....	15
4.3.2	Main Layout .....	19
4.3.3	Menus and Navigation .....	20
<b>5</b>	<b>Creating Plugins</b> .....	<b>22</b>
5.1	MASTER PLUGINS.....	22
5.1.1	Hello World .....	23
5.1.2	Overridable Methods .....	25
5.1.3	Working with Document and View .....	31
5.2	DOCUMENT PLUGINS .....	38
5.2.1	Hello World .....	39
5.2.2	Overridable Methods .....	42
5.3	EFILE PLUGINS .....	52
5.3.1	Technical Definition of an eFile.....	52
5.3.2	Hello World .....	54
5.3.3	Overridable Methods .....	57
<b>6</b>	<b>Usable Framework Components</b> .....	<b>61</b>
<b>7</b>	<b>Base Services and Components</b> .....	<b>62</b>
7.1	CONFIGURATION AND SETTINGS .....	62
7.1.1	Static Configuration .....	62
7.1.2	Dynamic Configuration .....	63
7.1.3	Creating Own Setting Controls .....	69
7.2	UI SYSTEM.....	73
7.2.1	Views and Layouts.....	73
7.2.2	Context.....	77
7.2.3	Designs .....	78
7.2.4	Menus and Navigation .....	78
7.2.5	Dialogs .....	86
7.2.6	Standard Views and Dialogs.....	89
7.2.7	Clipboard.....	93

7.2.8	Keyboard Shortcut Support .....	93
7.2.9	Call Scripts at onLoad.....	94
7.2.10	Sending HTML Form Data .....	95
7.2.11	Context Help.....	98
7.2.12	Localization.....	100
7.3	EXCEPTION HANDLING AND LOGGING .....	102
7.3.1	Standard Exceptions.....	103
7.3.2	Logging .....	104
7.4	ECM SYSTEM .....	106
7.4.1	ECM Model .....	107
7.4.2	DMSID and References .....	107
7.4.3	Application Objects .....	113
7.4.4	Search in the ECM Sys .....	114
7.4.5	Creating Objects .....	125
7.4.6	Native Access .....	125
7.4.7	Specifying Permissions .....	126
<b>8</b>	<b>Extended Components.....</b>	<b>130</b>
8.1	FIELD MANAGER.....	130
8.1.1	Creating Controls or New Field Managers .....	130
8.1.2	Client-Side JavaScript Validation.....	130
8.1.3	Combobox rendering .....	131
8.2	HISTORY MANAGER .....	132
8.2.1	Adding Plugin Events to the Audit Trail .....	132
8.3	MIME TYPE MANAGER .....	133
8.4	PAGING NAVIGATION COMPONENT .....	133
8.4.1	Implementation Details .....	133
8.4.2	Configuration details .....	134
8.4.3	Further Extensions.....	135
<b>9</b>	<b>Advanced Customizations.....</b>	<b>136</b>
9.1	SUPPORT OF ACCESSIBILITY .....	136
9.2	SUPPORT OF ADDITIONAL LANGUAGES .....	139
9.3	CONNECTING THIRD-PARTY-SYSTEMS .....	139
9.3.1	Remote control and deep links .....	139
9.3.2	Create and Extend Role Managers .....	144
9.4	CUSTOMIZING THE VIEWER .....	145
9.4.1	Creating a New Viewer Servlet.....	145
9.4.2	Modifying the IBM P8 and IS Viewer .....	145
9.5	INFORMATION ABOUT DISTRIBUTED DEPLOYMENT .....	147
9.5.1	Guaranteed Interoperability with Future Releases .....	148
<b>10</b>	<b>Problem Solving .....</b>	<b>149</b>
10.1	CONFIGURATION DUMP .....	149
10.2	REQUEST DUMP .....	150
<b>11</b>	<b>List of Figures .....</b>	<b>151</b>
<b>12</b>	<b>List of Tables.....</b>	<b>152</b>
<b>13</b>	<b>List of Code Fragments.....</b>	<b>153</b>
<b>14</b>	<b>Bibliography .....</b>	<b>156</b>



# 1 Introduction

## 1.1 Intended Audience

Readers of this document are assumed to have good knowledge of Java and JSP programming as well as the administration of an application server (needed for deploying OpenWorkdesk). If you want to change the layout, you need knowledge of CSS.

**⚠ Note: The prerequisite is that you know how to configure the components described in the Configuration Guide.**



## 1.2 About this Document

This manual describes how to develop plugins using the WeWebU OpenECM-Framework with the help of simple tutorials.

Chapter **Overview Framework** describes the architecture and all main components of OpenECM-Framework briefly. That is why this chapter can be used as quick reference. But you can also start right away with chapter **First Installation** and get a first impression of the framework with the test configuration (dummy ECM adapter). Customisation of the layout is described in chapter **Customizing Layout and Designs**. How to create plugins is explained in chapter **Creating Plugins**, chapter **Usable Framework Components** and chapter **Base Services and Components**. Chapter **Extended Components** specifies field, audit trail and MIME Type Manager components. Chapter **Advanced Customizations** describes how to connect to other systems and how to organize distributed deployment. At the end of the manual you find the section **Problem Solving**.

In addition, several supplementary chapters about special components (e.g. inbox plugin) are available on request.

This document should support you in developing plugins using the WeWebU OpenECM-Framework. We would be grateful to receive your feedback, questions or remarks at [contact@wewebu.de](mailto:contact@wewebu.de) in order to continuously improve it – thank you!



## 2 Overview Framework

### 2.1 Architecture

The OpenECM-Framework is based on a multitier architecture (see Fig. 1). The architecture is described in detail in [CONFPLAN, chapter 3.1 Architecture]

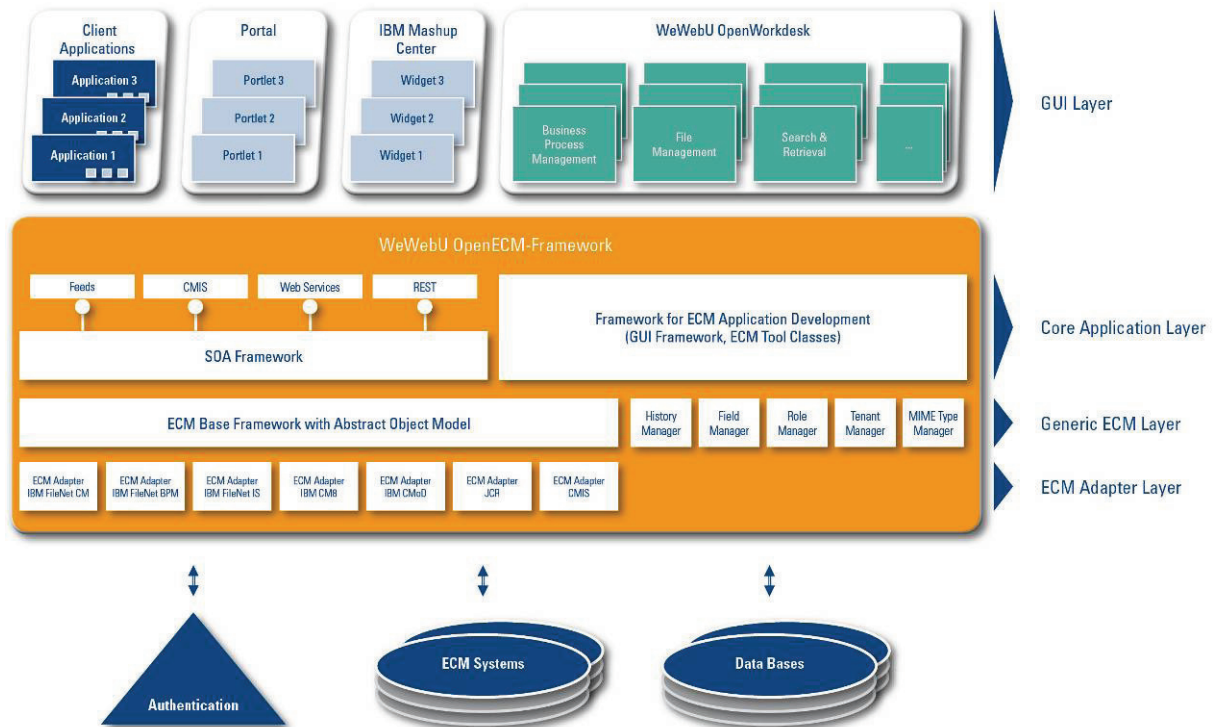


Fig. 1) Overview OpenECM-Framework architecture

The framework consists of the following components:

#### Master Plugins:

Master plugins undertake the main tasks of an application (e.g. retrieval).

#### Document Plugins:

Document plugins undertake sub-functions for processing objects (e.g. delete a document).

#### eFile Plugins:

eFile plugins undertake functions that can be applied to an eFile or at folder level (e.g. add documents).

#### Views and Dialogs:

Reusable components for displaying and processing of ECM objects (e.g. result lists)

**Clipboard:**

Can be called from all plugins to paste objects copied by a user

**Settings:**

Component to manage personalized settings that are available for each plugin

**Configuration:**

Component storing the whole configuration information of the framework (e.g. description of all plugins and start parameters)

**MIME Type Manager:**

The MIME Type Manager defines how objects are handled during processing. For each MIME type you can configure in the MIME table ([owmimetable.xml](#), [mimetypes.properties](#)) how it should be processed and which icon is assigned to it.

**History Manager:**

The History Manager logs events taking place either in the application or in the ECM system. Different History Managers can be chosen in the configuration plugin.

**Field Manager:**

The Field Manager displays and validates metadata of ECM objects. Different Field Managers can be chosen in the configuration plugin.

**Role Manager:**

The Role Manager defines which plugins and which layouts are available to a user after login. Different Role Managers can be chosen in the configuration plugin.

**Tenant Manager:**

The tenant manger tailors the configuration according to the mandator that is accessing the application.

**ECM Object Model:**

The ECM object model and the ECM adapter allow transparent access to the connected ECM system. Different ECM adapters can be chosen in the configuration plugin.





## 5 Creating Plugins

When you create a specific application, you can always try to find a solution with and implement it with the plugins on hand. But nevertheless you can also create your own plugins. New OpenECM-Framework plugins are interoperable with new releases, this means that plugins can also be used with future versions of WeWebU OpenECM-Framework. A new framework version does not override or modify your application.

This is achieved by using interfaces all plugins must implement, detailed version information in plugins and interfaces allowing verifying and checking a configuration's validity. (More information in chapter [9.5.1 Guaranteed Interoperability with Future Releases](#) and chapter [10.1 Configuration Dump](#).)

There are three types of plugins:

- master plugins for main tasks (e.g. retrieval)
- document plugins for functions on object level (document, folder, workflow or custom objects) (e.g. delete object)
- eFile plugins for functions on record level (e.g. add object)

In addition, OpenECM-Framework uses further plug-able components which are interoperable with future releases (e.g. Field Manager). These components are explained in chapter [8 Extended Components](#). The next three chapters describe how you create these three plugin types. Following, chapter [6 Usable Framework Components](#) explains how you can use the services and components of the framework for the plugins of your applications.

### 5.1 Master Plugins

Master plugins undertake a main task of the application (e.g. retrieval). A master plugin has a title and a display area which are displayed on activation. Master plugins are loaded after log-in. For each entry in the plugin descriptor, a plugin instance is created for each user session.

Master plugins implement the classes `OwMasterDocument` and `OwMasterView` and thus provide an optionally-available document / view pattern for the developer.

**⚠ The master plugin document class cannot be compared with documents from the ECM system. In fact, it takes over similar tasks as the model in the MVC pattern.**

Advantages:

- The document / view pattern extends the MVC pattern. With the MVC pattern it is only possible to send updates to all views. The document / view pattern makes it possible to send granular updates to specific document / view objects. Thus, it is possible to update only the views needing to be updated.
- Instances are always in the session context:
  - Thus, implementation is as easy as implementing a fat client.
  - Plugins only have to be initialized once after log-in. After this, they are constantly available for any further request.
  - Views, documents and other master plugins can be called directly for each instance. This leads to robust communication which can be verified at compile time.
- The update takes places as qualified update; this means that different events are generated in order to exchange information between views and documents.
- Moreover, the document / view pattern allows implementing own front controller.

### 5.1.1 Hello World

In order to help you to understand the master plugin concept, a simple master plugin is described below. Please create the following master plugin document class and the corresponding view class:

```
package com.wewebu.ow.server.plugin.owdemo.owhelloworld;

import com.wewebu.ow.server.app.*;

public class OwHelloWorldDocument extends OwMasterDocument
{
// === versioning
  /** minor version number of the used Interface */
  public static int INTERFACE_MINOR_VERSION          = 0;
  /** major version number of the used Interface */
  public static int INTERFACE_MAJOR_VERSION          = 1;
  /** update version number of the implementation */
  public static int IMPLEMENTAION_UPDATE_VERSION     = 0;
  /** minor version number of the implementation */
  public static int IMPLEMENTAION_MINOR_VERSION      = 0;
  /** major version number of the implementation */
  public static int IMPLEMENTAION_MAJOR_VERSION      = 1;

// document class can be empty

}
```



As you can see, you do not have to implement anything in the document. But it is helpful to add the version information to the **OwMasterDocument** and **OwMasterView** classes. You can then view them with the configuration dump (chapter 10.1).

```
package com.wewebu.ow.server.plugin.owdemo.owhelloworld;

import com.wewebu.ow.server.app.*;

import java.io.Writer;

public class OwHelloWorldView extends OwMasterView
{
// === versioning
    /** minor version number of the used Interface */
    public static int INTERFACE_MINOR_VERSION        = 0;
    /** major version number of the used Interface */
    public static int INTERFACE_MAJOR_VERSION        = 1;
    /** update version number of the implementation */
    public static int IMPLEMENTAION_UPDATE_VERSION    = 0;
    /** minor version number of the implementation */
    public static int IMPLEMENTAION_MINOR_VERSION    = 0;
    /** major version number of the implementation */
    public static int IMPLEMENTAION_MAJOR_VERSION    = 1;

    /** called when the view should create its HTML content to be displayed
    */
    protected void onRender(Writer w_p) throws Exception
    {
        w_p.write("Hello World.");
    }
}
```



**Code 3)** For this example, only the onRender() method which outputs text has been implemented.

You have to add the plugin in the plugin descriptor, otherwise OpenECM-Framework will not execute it. That is why you have to add the following section in **owplugins.xml**:

```
<PlugIn type="ow_master">
  <Name>Hello World</Name>
  <Description>Demoplugin</Description>
  <id>com.wewebu.ow.HelloWorld</id>
  <Vendor>WeWebU Software AG</Vendor>
  <Version>1.0.0</Version>
```

```

<ClassName>com.wewebu.ow.server.plugin.owdemo.owhelloworld.OwHelloWorldDocument
</ClassName>
<ViewClassName>com.wewebu.ow.server.plugin.owdemo.owhelloworld.OwHelloWorldView
</ViewClassName>
</PlugIn>

```

Code 4) Entry in plugin descriptor

**⚠ Restart the application server and test the plugin. After log-in a new master plugin with the title "HelloWorld" appears. Clicking on this plugin, the text "HelloWorld" is shown.**

## 5.1.2 Overridable Methods

### 5.1.2.1 Initialize

View as well as document class have an overridable `init()` method that is called after creating and initializing the plugin. The core system and all main components are available when the `init()` method is called; i.e. you can already access the ECM system and the context at this point. Please keep in mind that unlike in common MVC models the `init()` method is only called once at log-in. Normally, you create further views or initialize resources which the plugin needs in the `init()` method.

```

/** init the view after the context is set.
 */
protected void init() throws Exception
{
    super.init();

    // your own initialisation code...
}

```



Code 5) init Method

**⚠ Call the super method as well because the core system may need it for carrying out necessary initializations.**

### 5.1.2.2 Activation

If you click a master plugin, it is activated, i.e. it appears in the front and the previous plugin disappears. You can override the `onActivate()` method in `OwMasterView` if you want to be

displayed after the request was executed. If in the SQL statement a column needed to process the request is missing, the SQL parser of the database will throw/return an error.

If the newly created document function is using some properties not retrieved during the main request, the OpenECM performs requests to the backend dynamically to get the missing information.

**⚠ This failover handling enables retrieving the information on the fly, without additional source code to be programmed by the developer. But at the same time this could create a performance issue, if thousand of documents were requested.**

```
public Collection getRetrievalPropertyNames() throws Exception
```

**Code 38)** Method to reduce ECM requests

By default this method returns null if no additional metadata is needed by the document.

**⚠ If the document function uses metadata not retrieved during the “main” request to the ECM system, the performance can be decreased to a tenth of the normal execution time.**

## 5.3 eFile plugins

eFile plugins are functions on eFile level (e.g. add an object). eFile plugins are similar to document plugins in their structure but they work in a different context. Document plugins work with any objects (documents as well as folders or eFiles). eFile plugins only work with eFiles.

**⚠ Document plugins are displayed nearly everywhere in OpenWorkdesk where objects are used. In contrast, eFile plugins are only used in the master plugin eFile management.**

### 5.3.1 Technical Definition of an eFile

An eFile is a folder object with different subfolders. A eFile plugin can execute actions on the top level folder as well as on the currently selected subfolder. Both objects are passed in the `onClickEvent()` method.



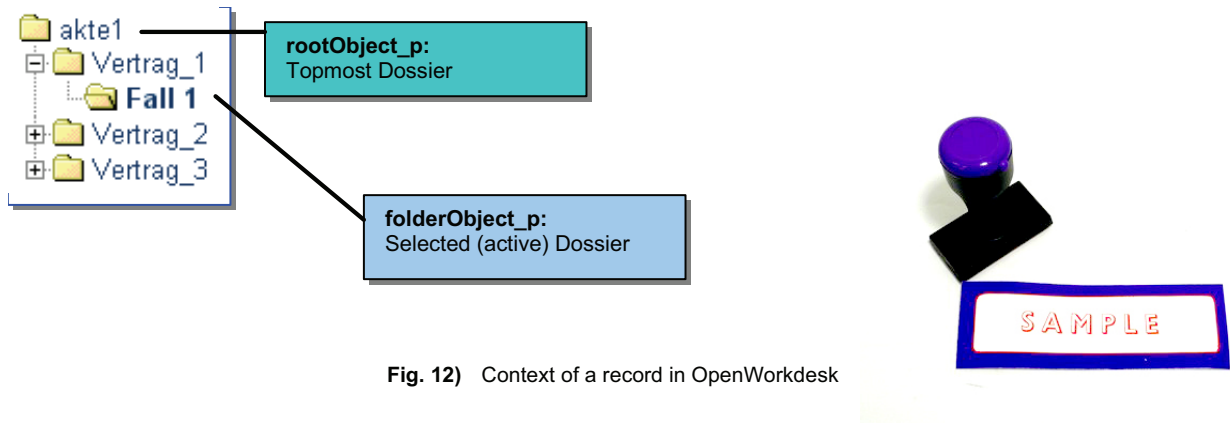


Fig. 12) Context of a record in OpenWorkdesk

A eFile plugin has a label, an icon, a tool tip and a click function. The click function can only be executed on single folders selected in file management. Label and icon can be dynamically generated at runtime out of the file properties. eFile plugins are loaded after log-in. Exactly one plugin instance is generated for each user session per entry in the plugin descriptor. If the same eFile plugin is used on several objects, the same plugin instance is always used.

eFile plugins extend class: `com.wewebu.ow.server.app.OwRecordFunction`.

A eFile plugin is registered in the plugin descriptor with the following section:

```
<PlugIn type="ow_recordfunction">
  <!--name which is displayed as label -->
  <Name>[PluginName]</Name>

  <!--description which is also displayed as tool tip -->
  <Description>[description]</Description>

  <!--unique ID with which the plugin is registered -->
  <id>[ID]</id>

  <!--vendor -->
  <Vendor>[Herstellername]</Vendor>

  <!--plugin version -->
  <Version>[triple digit version number]</Version>

  <!--(optional) context help -->
  <helppath>
    [path to JSP help page below /help_<locale>]</helppath>

  <!--class name -->
```

```

<ClassName>[fully specified Java class name]</ClassName>

<!--list of object class names (of the selected subfolder) for which the plugin
is activated, an empty list activates all classes -->
<objectclasses>
  <name>[ObjectClassName]</name>
</objectclasses>

</PlugIn>

```

Code 39)

Registration of a eFile plugin in the plugin de



SAMPLE

## 5.3.2 Hello World

In order to help you to understand the concept, a simple eFile plugin is described below. Please create the following eFile plugin class:

```

package com.wewebu.ow.server.plugin.owdemo.owhelloworld;

import com.wewebu.ow.server.app.*;
import com.wewebu.ow.server.util.*;
import com.wewebu.ow.server.ecm.*;

public class OwHelloWorldRecordFunction extends OwRecordFunction
{
// === versioning
  /** minor version number of the used Interface */
  public static int INTERFACE_MINOR_VERSION          = 0;
  /** major version number of the used Interface */
  public static int INTERFACE_MAJOR_VERSION          = 1;
  /** update version number of the implementation */
  public static int IMPLEMENTAION_UPDATE_VERSION     = 0;
  /** minor version number of the implementation */
  public static int IMPLEMENTAION_MINOR_VERSION      = 0;
  /** major version number of the implementation */
  public static int IMPLEMENTAION_MAJOR_VERSION      = 1;

// === members

  /** set the plugin description node
   * @param Node_p OwXMLUtil wrapped DOM Node containing the plugin description
   * @param Context_p OwMainAppContext

```

```

    */
    public void init(OwXMLUtil Node_p,OwMainAppContext Context_p) throws Exception
    {
        super.init(Node_p,Context_p);

        // set icon url
        m_strIconURL = "[ICON]";
    }

    /** event called when user clicked the plugin label / icon
    *
    * @param rootObject_p OwObject root folder to work on
    * @param folderObject_p OwObject selected folder to work on
    * @param refreshCtx_p OwFunctionRefreshContext callback interface for the
    function plugins to signal refresh events to clients, can be null if no refresh is
    needed
    *
    * @return boolean true = refresh the record, false = no refresh is necessary
    */
    public void onClickEvent(OwObject rootObject_p, OwObject folderObject_p,
OwClientRefreshContext refreshCtx_p) throws Exception
    {
        // display message box with the name of the selected object
        getContext().openDialog(
            new OwMessageBox(
                OwMessageBox.TYPE_OK,
                OwMessageBox.ICON_TYPE_INFO,"Hello World","Akte: " +
rootObject_p.getName() + " Unterordner: " + folderObject_p.getName()),
                null
            );
    }
}

```



**Code 40)** Creating a eFile plugin class

It is helpful to directly add the version information to **OwRecordFunction** class so that you can then request it with the configuration dump (chapter 10.1)

In the `init()` method only the icon member is initialized with text. Normally, you define an `<img...>` HTML tag here but you can also just render the text. Alternatively, it is also possible to override the `getIconHTML()` method directly and even to define which icon is shown depending on the file status. You find more information about this below. Finally, the `onClickEvent()` method opens a message

box (see chapter 6 Usable Framework Components) and shows the title of the selected eFile and subfolder.

You have to add the plugin to the plugin descriptor because otherwise OpenECM-Framework will not execute it. Please add the following section at the end of **owplugins.xml**:

```
<PlugIn type="ow_recordfunction">
  <Name>Hello World</Name>
  <Description>Vorinstalliertes Plugin von WeWebU Software AG.</Description>
  <id>com.wewebu.ow.owdemo.owhelloworldrecordfunction</id>
  <Vendor>WeWebU Software AG</Vendor>
  <Version>1.0.0</Version>

  <ClassName>com.wewebu.ow.server.plug.owdemo.owhelloworld.OwHelloWorldRecordFunction</ClassName>

</PlugIn>
```

Code 41)

Entry of the eFile plugin in plugin descriptor

- ⚠ **Restart the application server and test the plugin. After log-in, start a search in the default configuration and open a record. You will then see a new eFile plugin with the title "HelloWorld" in the record management. Clicking it, a dialog is opened showing the name of the selected eFile and subfolder.**



Fig. 13) Hello World eFile plugin with open dialog



## 8 Extended Components

This chapter describes further services and gives an overview over th

### 8.1 Field Manager

The Field Manager is the central component that specifies the display and behavior of metadata in views, lists and dialogs. This means that each time metadata is displayed it is rendered via the Field Manager.

You can write your own Field Manager and thus specify the behavior of certain metadata or you can write controls for certain metadata types that are then configured for the Field Manager in the bootstrap file. The standard implementation (`OwStandardFieldmanager`) of the Field Manager reads these configuration data from the `owbootstrap.xml`, loads and initializes the corresponding controls.

#### 8.1.1 Creating Controls or New Field Managers

Field Manager controls implement the class `OwFieldManagerControl`. They must then implement the methods `insertReadOnlyField()`, `insertEditField()` and `updateField()`. The field control can be registered in `owbootstrap.xml` with a metadata or a Java class name with which it is then called.

The method `insertReadOnlyField()` is called if a property is displayed as read-only field. Normally, the output of the value as text is sufficient here.

The method `insertEditField()` is called if a property is displayed as editable. Here you render a corresponding HTML form element. You use the parameter `strID_p` as ID.

Finally, the method `updateField()` is called if the user stores modified data. Here you read the modified value from the request parameter with the help of the parameter ID and return it. If the input is wrong, you can throw an exception that is displayed next to the property.

#### 8.1.2 Client-Side JavaScript Validation

The Field Manager validates content provided by the user after it has been sent to the server. To provide hints on invalid field contents while the user fills out a form, you can add JavaScript code to the function `onCustomValidation()` in the file `js/common.js`. This function gets called with the field class (e.g. the property name) and the current value whenever the user leaves an input field. If the current field value is valid for that metadata, this function has to return null. Otherwise, this function has to return a string with the message for the user.

```
function
onCustomValidation(classname,javaclassname,fieldprividertype,fieldprovidertype,fieldid,value)
{
    if(classname=="ow_Filename"){
        if(value.indexOf('!')==0){
            return(" \!" is not allowed as first character! The name has to start
                with a letter or a number.");
        }
        else{
            return(null);
        }
    }
    return(null);
}
```

Code 122)

Example: Client-Side javascript validation



The OpenECM Framework will deal with the result and present the error message to the user by inserting it into an empty HTML element.

Property	Value
Name:	Adresse "!" is not allowed as first character! The name has to start with a letter or a number.

Fig. 27) Example: Rendered error message beside a text field

### 8.1.3 Combobox rendering

It is recommendable to use the component `OwComboboxRenderer` for combobox rendering. This component has the ability to render comboboxes as classic HTML combos, or as EXTJS components, depending by the implementation class name specified in `<ComboboxRendererClassName/>` element from `owbootstrap.xml` configuration file.

The items are provided to the renderer component as an instance of `OwComboModel` interface.

The component has the ability to receive and render different CSS style classes, and support different JavaScript events and event handlers.

Example:

```
List recentRoleItems = new LinkedList();
List recent = m_View.getRecentRoleNames();
for (int i = recent.size() - 1; i >= 0; i--)
{
    String recentRoleName = (String) recent.get(i);
    OwComboItem item = new OwDefaultComboItem("" + i,
recentRoleName);
    recentRoleItems.add(item);
}
```

```

    }
    OwComboModel recentRolesModel = new OwDefaultComboModel(false,
false, m_View.getCurrentRoleName(), recentRoleItems);
    OwComboboxRenderer renderer = ((OwMainAppContext)
m_View.getContext()).createComboboxRenderer(false, recentRolesModel,
"owrolerecent", null, null);
    renderer.addEvent("onchange", "changeToRecentRole()");

```

The example above shown how the component can be configured with items, and how to attach a JavaScript event to the component.. Events have different names on Javascript and in EXTJS, and differents paramters, so is strongly recommended that event handlers should be javascript functions with empty parameters list.

## 8.2 History Manager

The audit trail is represented with the History Manager - a replaceable component which can be configured with owbootstrap.xml. You can enter your own events to the audit trail as follows:

```

OwEventManager events = ((OwMainAppContext)getContext()).getHistoryManager();

events.addEvent(OwEventManager.HISTORY_EVENT_TYPE_GENERIC,OwEventManager.
HISTORY_EVENT_ID_OBJECT_MODIFY_PROPERTIES,null,OwEventManager.HISTORY_STATUS_OK);

```

**Code 123)**

Example: add event to audit trail

For further information please refer to the JavaDoc.



### 8.2.1 Adding Plugin Events to the Audit Trail

If you only want to add plugin events, you can use the already implemented event method `addHistoryEvent()`.

```

addHistoryEvent(rootObject_p, folderObject_p, OwEventManager.HISTORY_EVENT_TYPE_
PLUGIN_INVOKE_EDIT, OwEventManager.HISTORY_STATUS_OK);

```

**Code 124)**

Example for eFile plugins

```

addHistoryEvent(Objects_p, oParent_p, OwEventManager.HISTORY_EVENT_TYPE_PLUGIN_
INVOKE_EDIT, OwEventManager.HISTORY_STATUS_OK);

```

**Code 125)**

Example for document plugins

## 13 List of Code Fragments

<b>Code 1)</b>	Rendering of regions with renderRegion().....	18
<b>Code 2)</b>	Checking whether Region exists .....	19
<b>Code 3)</b>	For this example, only the onRender() method which outputs text has been implemented. ....	24
<b>Code 4)</b>	Entry in plugin descriptor .....	25
<b>Code 5)</b>	init Method .....	25
<b>Code 6)</b>	onActivate() method.....	26
<b>Code 7)</b>	on Render() method.....	26
<b>Code 8)</b>	Header for JSP pages .....	27
<b>Code 9)</b>	OwHelloWorldView class.....	29
<b>Code 10)</b>	OwHelloWorldDocument class .....	29
<b>Code 11)</b>	JSP page OwHelloWorldView.jsp.....	30
<b>Code 12)</b>	You send an update event with OwDocument.update() .....	32
<b>Code 13)</b>	OwView.onUpdate() method.....	32
<b>Code 14)</b>	broadcast() method.....	33
<b>Code 15)</b>	OwView.setDocument() method .....	33
<b>Code 16)</b>	Example for distinguishing events .....	34
<b>Code 17)</b>	Identifying the document class of a master plugin.....	35
<b>Code 18)</b>	Call of dispatch method with OwMasterDokument.dispatch().....	36
<b>Code 19)</b>	Catching of dispatch calls when overriding OwMasterDokument in onDispatch().....	36
<b>Code 20)</b>	Obligatory entries in plugin descriptor for document plugin.....	39
<b>Code 21)</b>	Creating example document function class .....	40
<b>Code 22)</b>	Entry in plugin descriptor .....	41
<b>Code 23)</b>	init() method.....	42
<b>Code 24)</b>	onClickEvent() method .....	43
<b>Code 25)</b>	onClientRefreshContextUpdate() method.....	44
<b>Code 26)</b>	Assigning an icon to a document plugin .....	46
<b>Code 27)</b>	Specifying a label for a document plugin .....	46
<b>Code 28)</b>	Specifying a tool tip for a document plugin .....	46
<b>Code 29)</b>	Dynamic generation of icons and labels depending on the selected object 1 .....	47
<b>Code 30)</b>	Dynamic generation of icons and labels depending on the selected object 2 .....	48
<b>Code 31)</b>	Activation of document plugins .....	48
<b>Code 32)</b>	Utility method for filtering a plugin against certain object types and object classes .....	49
<b>Code 33)</b>	Specifying filter criteria.....	49
<b>Code 34)</b>	Checks whether a parent object exists .....	50
<b>Code 35)</b>	Creating an audit trail for single objects.....	50
<b>Code 36)</b>	Creating an audit trail for multiple objects.....	51
<b>Code 37)</b>	Adding events to the audit trail .....	51
<b>Code 38)</b>	Method to reduce ECM requests .....	52
<b>Code 39)</b>	Registration of a eFile plugin in the plugin descriptor.....	54
<b>Code 40)</b>	Creating a eFile plugin class.....	55
<b>Code 41)</b>	Entry of the eFile plugin in plugin descriptor.....	56
<b>Code 42)</b>	init() method in the eFile plugin.....	57
<b>Code 43)</b>	onClickEvent() method in the eFile plugin .....	57
<b>Code 44)</b>	Set up of filters in eFile plugin.....	58
<b>Code 45)</b>	Utility function for filtering a plugin against certain object classes .....	59
<b>Code 46)</b>	Specifying of filter criteria in the eFile plugin .....	59
<b>Code 47)</b>	Audit trail of eFile plugin .....	60
<b>Code 48)</b>	Adding events to the audit trail .....	60
<b>Code 49)</b>	Entry of XML node "MyIntegerValue" .....	62
<b>Code 50)</b>	Call of value in plugin.....	62
<b>Code 51)</b>	Access to further subnodes .....	63
<b>Code 52)</b>	Accessing dynamic settings.....	63
<b>Code 53)</b>	Plugin descriptor section with dynamic settings in settingsset .....	64



<b>Code 54)</b>	Example: Modifying a color property with the property control .....	66
<b>Code 55)</b>	Predefining colors .....	66
<b>Code 56)</b>	Property's definition in plugin descriptor .....	67
<b>Code 57)</b>	getBGColor() method.....	68
<b>Code 58)</b>	Overridable functions from OwSettingsPropertyBaseImpl 1.....	70
<b>Code 59)</b>	Overridable functions from OwSettingsPropertyBaseImpl 2.....	71
<b>Code 60)</b>	Overridable functions from OwSettingsPropertyBaseImpl 3.....	71
<b>Code 61)</b>	Overridable functions from OwSettingsPropertyBaseImpl 4.....	71
<b>Code 62)</b>	Overridable functions from OwSettingsPropertyBaseImpl 5.....	72
<b>Code 63)</b>	Overridable functions from OwSettingsPropertyBaseImpl 6.....	72
<b>Code 64)</b>	Configuration of a property control .....	73
<b>Code 65)</b>	Creating and embedding a new view.....	74
<b>Code 66)</b>	Embedding other views.....	74
<b>Code 67)</b>	Creating events.....	75
<b>Code 68)</b>	Signature of event method.....	75
<b>Code 69)</b>	OwLayout class.....	75
<b>Code 70)</b>	Passing rendering to a JSP page .....	77
<b>Code 71)</b>	Context methods for requesting design resources .....	78
<b>Code 72)</b>	Example: Adding an image.....	78
<b>Code 73)</b>	Easily react on events with the reason parameter.....	80
<b>Code 74)</b>	Allowed addMenuItem methods .....	80
<b>Code 75)</b>	Registration of views to the navigation bar – OwSubNavigationView class .....	81
<b>Code 76)</b>	onActivate() method.....	81
<b>Code 77)</b>	Code Example 1 .....	83
<b>Code 78)</b>	Code Example 2 .....	84
<b>Code 79)</b>	Code Example 3 .....	85
<b>Code 80)</b>	Code Example 4 .....	85
<b>Code 81)</b>	Call to open a dialog .....	86
<b>Code 82)</b>	Callback method when closing a dialog.....	87
<b>Code 83)</b>	open a new dialog.....	87
<b>Code 84)</b>	Code Example 1 .....	88
<b>Code 85)</b>	Code Example 2 .....	88
<b>Code 86)</b>	Code Example 3 .....	89
<b>Code 87)</b>	Shortcut support.....	94
<b>Code 88)</b>	Calling of scripts at onLoad Event .....	94
<b>Code 89)</b>	Parallel usage of multiple views with form data .....	98
<b>Code 90)</b>	Specifying the JSP context help page in the plugin section of the plugin descriptor.....	98
<b>Code 91)</b>	Header for context help page .....	99
<b>Code 92)</b>	Adding an image.....	100
<b>Code 93)</b>	Creating a hyperlink to another help page.....	100
<b>Code 94)</b>	Generating localized text .....	100
<b>Code 95)</b>	Localization methods in the context OwApplicationContext .....	101
<b>Code 96)</b>	Example for specifying placeholders .....	102
<b>Code 97)</b>	Text displayed with this method in the application.....	102
<b>Code 98)</b>	Throwing the Cause Exception .....	104
<b>Code 99)</b>	Error Message Example .....	104
<b>Code 100)</b>	Localization of exceptions .....	104
<b>Code 101)</b>	Logger for the core package .....	105
<b>Code 102)</b>	Logger for other packages than OWD.....	105
<b>Code 103)</b>	Formatting logging messages .....	105
<b>Code 104)</b>	Logging messages 1 .....	106
<b>Code 105)</b>	Logging messages 2 .....	106
<b>Code 106)</b>	Access to the ECM system .....	106
<b>Code 107)</b>	Code example 1 .....	111
<b>Code 108)</b>	Code example 2 .....	112
<b>Code 109)</b>	Accessing application data.....	113
<b>Code 110)</b>	Code Example 1 .....	119



<b>Code 111)</b>	Code Example 2 .....	119
<b>Code 112)</b>	Code example 3 .....	120
<b>Code 113)</b>	Manually creating a search tree .....	124
<b>Code 114)</b>	Creating Objects .....	125
<b>Code 115)</b>	Accessing the native BaseObject of an OpenECM object .....	125
<b>Code 116)</b>	Getting the IBM FileNet P8 session object .....	126
<b>Code 117)</b>	Native query against P8 .....	126
<b>Code 118)</b>	Configuring of programmatic permission to P8 objects .....	127
<b>Code 119)</b>	Activating the permission dialog in the edit properties plugin.....	128
<b>Code 120)</b>	Creating dialog class with OwObjectAccessRightsView .....	129
<b>Code 121)</b>	Open dialog .....	129
<b>Code 122)</b>	Example: Client-Side javascript validation .....	131
<b>Code 123)</b>	Example: add event to audit trail .....	132
<b>Code 124)</b>	Example for eFile plugins .....	132
<b>Code 125)</b>	Example for document plugins .....	132
<b>Code 126)</b>	Customizing a paging navigator .....	135
<b>Code 127)</b>	Entry of plugin version in XML descriptor .....	148



If you want to read more and want to get the most out of your OpenWorkdesk installation go to <http://www.openworkdesk.org/> and register by clicking at the "REGISTER" button on the top right side of the page. Afterwards you can download the full version of this document and other documentation from <http://www.openworkdesk.org/guides> . Furthermore you can post comments and questions in the forum at <http://www.openworkdesk.org/phpbbforum> . Give it a try!

