



CORDIS

CORDIS CMS2 V4

Functional Specification Document

Contents

1	Introduction	3
1.1	Scope of the Document.....	3
1.2	Definitions, Acronyms Abbreviations.....	3
1.3	References	4
2	Functional Specification	4
2.1	Modifications not related to User Requirements	4
2.1.1	<i>Update Plone to Version 2.5.3</i>	4
2.1.2	<i>Update Plone to Version 3</i>	5
2.1.3	<i>Update existing Content Types to use Interfaces and GenericSetup</i>	5
2.1.4	<i>Unify SimpleDocument and Text</i>	6
2.1.5	<i>Unify SimpleDocument and Composed Document</i>	6
2.1.6	<i>Make all text content related fields virtual</i>	7
2.1.7	<i>Use lxml library throughout CMS2 for Handling of XML/HTML</i>	7
2.1.8	<i>Composition as External Service</i>	8
2.1.9	<i>PAS-Adapter to SUR-Service</i>	8
2.1.10	<i>Preparation for ICA2 backend integration</i>	9
2.1.11	<i>Include LinguaPlone internationalization Product for Plone</i>	9
2.1.12	<i>Enable Email Notification for Workflow Transitions</i>	10
2.2	General Modifications and Extensions.....	11
2.2.1	<i>General File Upload instead of only Zip Files</i>	11
2.2.2	<i>Check uploaded Documents before Object Creation</i>	12
2.2.3	<i>Import Time and Modifications after Import</i>	12
2.2.4	<i>Move, Rename, Copy, Paste for Workspace</i>	13
2.3	Multilingual Support.....	15
2.3.1	<i>ISO Language Code Extraction from Filename</i>	15
2.3.2	<i>Language dependent Versions of Resources</i>	15
2.3.3	<i>Language dependent Versions and Folder Listing</i>	15
2.3.4	<i>Resource Language Indicators</i>	16
2.3.5	<i>Language dependent Versions and Permissions</i>	16
2.3.6	<i>Language Dependent Versions and Composed Documents</i>	17
2.4	WYSIWYG Display and Editing.....	18
2.4.1	<i>WYSIWYG display of (X)HTML documents</i>	18
2.4.1.1	<i>Views for (X)HTML documents</i>	18
2.4.2	<i>WYSIWYG Editors and local Link Handling</i>	19
2.4.3	<i>WYSIWYG Editors and Markup Modification</i>	19
2.4.4	<i>WYSIWYG Editing and CSS Styles and Classes</i>	20
2.4.4.1	<i>WYSIWYG Editor that supports CSS Style Selection</i>	20
2.4.4.2	<i>CSS Style Selection in FCKeditor</i>	20
2.4.4.3	<i>FCKeditor Customisation</i>	21
2.4.4.4	<i>CSS Dropdown List Customization</i>	21
2.4.5	<i>Handle Document Type Setting for HTML pages</i>	22

2.4.6	<i>Active Archive for HTML pages</i>	22
2.4.7	<i>Search and Replace</i>	23
2.4.8	<i>Placeholder Insertion Button</i>	24
2.5	<i>Composed Documents</i>	25
2.5.1	<i>Visualization of Permissions on Components</i>	25
2.5.2	<i>Visualization of Components as Boxes or Frames</i>	25
2.5.3	<i>Checking Component Modifications upon Save</i>	26
2.6	<i>Other Content</i>	26
2.6.1	<i>Composed Document, XML Component and XSL Stylesheet</i>	27
2.6.2	<i>Dynamic Content, Queries, Search Abstraction Layer</i>	27
2.6.3	<i>Coldfusion Templates</i>	28
2.7	<i>Documentation</i>	29
2.7.1	<i>Online Help / FAQ</i>	29
2.7.2	<i>Documentation and Release</i>	29
2.7.3	<i>Content-Manual</i>	30
2.7.3.1	<i>WYSIWYG Editors and HTML Markup</i>	30
2.7.3.2	<i>ComposedEdit and HTML</i>	30
2.8	<i>Versioning</i>	31
2.8.1	<i>Central Workspace and Backup Copies of Resources</i>	31
2.9	<i>Users, Permission Handling and Authentication</i>	32
2.9.1	<i>CMSPowerUser Role</i>	32
2.9.2	<i>User Export and Import</i>	32
3	<i>Traceability Matrix</i>	33

1 Introduction

This document describes the functional aspects for the implementation of the User Requirements for CORDIS CMS2 Version 4 as described in the user requirements for version 4.

1.1 Scope of the Document

The user requirements as well as additional requirements implied by the implementation are described in section 2. A traceability matrix links the user requirement issues to the issues from the functional specification.

Each functional requirement also lists deliverables for the corresponding release.

1.2 Definitions, Acronyms Abbreviations

The following table represents the most significant terms used in this document

Term	Definition
ADS	Automatic Delivery System
CMS	Content Management System
Component	Content that can be reused in multiple documents
Composition	Constructing a document from different, reusable parts
CSS	Cascading Stylesheet Syntax
Template	document where other components will be inserted
Placeholder	A named location in a document, where a component can be inserted

1.3 References

Lot2_RSD_CMS2 V4_v040.doc

2 Functional Specification

2.1 Modifications not related to User Requirements

[Zope](#) and [Plone](#) (on which the CMS is based) are changing rapidly in order to make them and products based on them modular and easier to maintain. The current Zope2 is being rewritten based on new techniques resulting in Zope3 while Plone (currently at version 2.5.3) uses Zope3 technologies in its new version 3.0 while still using Zope2

The most important changes are related to maintainability and decoupling configuration information from code. This allows to use the code developed for the CMS to be more independent from the framework the CMS is running in ([Plone](#) in this case).

Fortunately, the new techniques for writing modular independent packages for [Plone](#) the Plone3 and Zope3 way can already be applied in Zope2 and Plone-2.5.* with the help of a (forward) compatibility package (called [Five](#) which stands for 2 + 3), so no big bang is required.

It is however important to adapt existing CMS products to the new technology so that new features from the new releases can be easily used and to enhance the maintainability of the CMS code base.

2.1.1 Update Plone to Version 2.5.3

The current version of Plone in the CMS is 2.5.2. Before upgrading the CMS to use Plone 3.0, it will be updated to the latest 2.5 release which is 2.5.3 at the time of this writing. For a smooth upgrade path, 2.5.3 should not be skipped.

FS_NUR_P25: Update Plone to Version 2.5.3

Update Plone to Version 2.5.3 and adjust the CMS to run with it.

Deliverables:

- include Plone 2.5.3 in the first Release for CMS Version 4

2.1.2 Update Plone to Version 3

On August 17th 2007, Plone 3.0 was released which is a major step in functionality. A lot of the enhancements in that release will also be useful for the CMS.

FS_NUR_P30: Update Plone to Version 3.0

Update Plone to Version 3.0 and adjust the CMS to run with it.

Deliverables:

- include Plone 3.0 in the second Release of CMS Version 4

2.1.3 Update existing Content Types to use Interfaces and GenericSetup

The content types for CORDIS (SimpleDocument, ComposedDocument, Text, Image, File and Folder) as well as the Workspace are based on technologies from Plone-2.1. They use multiple inheritance to also derive from so called MixinClasses to add independent functionalities together.

This however locks in the MixinClasses into the Plone world because the MixinClasses make implicit assumptions about attributes or fields in the classes that mix them in (e.g. the content types).

FS_NUR_IGS: Interfaces and GenericSetup

Replace the existing mixin classes for CORDIS content types by interfaces and implementations and bind them via GenericSetup.

Deliverables:

- independent content capabilities specified by interfaces
- existing mixin classes as implementations of interfaces
- GenericSetup based configuration of interfaces for content types

2.1.4 Unify SimpleDocument and Text

If a HTML page contains errors, it cannot be imported into the CMS for WYSIWYG editing, e.g. as a SimpleDocument. Instead it has to be imported as a normal Text object and edited inside a normal textarea in order to fix the errors.

Unfortunately it cannot be immediately transformed into a SimpleDocument for WYSIWYG editing once the error has been fixed. Instead it has to be delivered first before it can be imported into the CMS as a SimpleDocument.

By unifying SimpleDocument and Text, a HTML document can be edited in WYSIWYG mode directly once the errors have been fixed.

FS_NUR_TXT: Unified Text and Simple Document

Unify the Text and SimpleDocument content types into one common content type for text based content that can be edited inside the browser. If the mime type is error free HTML, WYSIWYG editing can be enabled.

Deliverables:

- one unified content type for text base content
- additional instance field for selecting editor
- WYSIWYG editing supported only for error free HTML

2.1.5 Unify SimpleDocument and Composed Document

After SimpleDocument and Text have been unified into a Document content type, Composed Document should be unified with Document too, as its external representation is XML (the internal representation is a number of fields).

When unified, a user (with corresponding permissions) can either use the fields (with reference browser popups and a datagrid) to manipulate a composed document or directly edit the XML.

In order to achieve that, all the currently existing fields will be made virtual by reading their values from or storing their values into the XML that is stored in the text field instead of keeping the values in individual fields.

A composed document can thus be created by creating a (text) document and selecting the text/cdd mimetype. The user can then either write the XML of the composed document manually or modify it with the corresponding virtual fields for selecting the template and filling the placeholder.

FS_NUR_TXT: Unified Text and Simple Document

Unify the Document and Composed Document content types into one common content type for text based content that can be edited inside the browser. If the mime type is text/cdd then additional (virtual) fields will be displayed when editing for selecting template and placeholder fillers.

Deliverables:

- one unified content type for text base content, HTML files and composed documents
- fields from the former composed document only displayed, when the mime type is text/cdd

2.1.6 Make all text content related fields virtual

All the fields in the CMS that help the user to modify specific parts of a text with a specific widget will be made virtual. This means that they read their value from the text (e.g. the meta fields in the head of a HTML file) and store them back to the text after the modification, e.g. the field values the user sees are not stored separately from the text any more.

FS_NUR_VTF: Virtual Text Fields

make all fields that are related to the content of a text virtual, except the text field itself

Deliverables:

- for each field define an accessor and a mutator that fetches / stores the value from/into the text

2.1.7 Use lxml library throughout CMS2 for Handling of XML/HTML

The lxml library is a python package for working with XML, HTML, XSLT and XPATH, using the efficient, stable and powerful c implementations of libxml and libxslt. Since version 2.0 (released in February 2008) lxml also provides strong support for parsing and handling HTML, e.g.

- successfully parsing HTML that has typical errors or is not XHTML compliant into an tree structure.
- providing methods for rewriting link resources in a lot of HTML tags like action, archive, background, cite, classid, codebase, data, href, longdesc, profile, src, usemap, dynsrc, or lowsrc and also style attributes for url(link), and <style> tags for @import and url().
- providing methods for detecting the diffences before and after a modification to HTML, ignoring invisible whitespace differences or different attribute orders in tags.

FS_NUR_LXML: lxml Library Usage

Use the lxml library exclusively for handling of HTML and XML.

Deliverables:

- use lxml exclusively for parsing and writing XML and HTML
- store XML or HTML in the ZODB but work on lxml tree representations of them internally to make it easier to make modifications to the content
- send HTML to the browser and convert to a lxml tree when it's sent back
- determine diffs for HTML with lxml and present the results to the user before saving modifications.
- rewrite composition to work on lxml trees instead of the ad hoc approach based on regular expressions.

2.1.8 Composition as External Service

The rendering for composed documents (composition) will be handled outside of the CMS for production, as the CMS is used for authoring static pages, but not for publishing. Currently, the CMS has its own implementation of composition for the preview of composed documents.

In order to support an external service for composition, some changes have to be made to the CMS to allow the external service easy access to the required information.

FS_NUR_EXTC: external composition

support an external composition service for the preview of composed documents.

Deliverables:

- when accessing a resource directly (e.g. without a view), return the raw content with the corresponding mimetype as the content type (this is already the case for images, css, javascript etc) but was not yet the case for e.g. composed documents.
- add a configuration option for the external composer so that the preview actions knows where to request the composition
- implement a simple server for composition and qry evaluation based on lxml parsing to demonstrate, that the external service works as expected.
- use the new simple composition code inside the CMS as a fallback for when the external service is not available or not configured.

2.1.9 PAS-Adapter to SUR-Service

Plone has a Pluggable Authentication System for writing plugins to external authentication services. The CMS authentication module to CORDIS has been developed before the [PAS](#) and the SUR where available.

The current solution has some drawbacks, e.g. authentication information about the user is stored in the CMS and thus duplicated and therefore not automatically updated when the user modifies this information in CORDIS.

As the authentication module has to be rewritten for SUR anyway it should be based on [PAS](#) which results in higher flexibility and no need for synchronisation as the information will only be stored in one place, inside the SUR service.

FS_NUR_PAS: Pluggable Authentication Adapter for SUR

Change the CMS authentication to work with the SUR service. Use the [PAS](#) as the basis for the implementation.

Deliverables:

- include [PAS](#) product in the release
- rewrite authentication adapter based on [PAS](#)
- prepare authentication module for CORDIS SUR service.

2.1.10 Preparation for ICA2 backend integration

The CORDIS web resources are currently served from the filesystem because they are not yet managed in a database. This will change when ICA2 will be in production. At that time, it will be easy to store additional metadata about the resources and index them.

In order to prepare the ICA2 transition for the backend store, the current filesystem integration for the CMS workspace should be made independent of the filesystem API and defined as an interface with the filesystem backend as an initial implementation with an ICA2 implementation to follow later.

FS_NUR_BIP: Backend Intergation Preparation for ICA2

Define an plugin interface for backend storages for the workspace inside the CMS and use that interface for the filesystem backend.

Deliverables:

- an interface for workspace storage access
- an implementation of the interface for filesystem access

2.1.11 Include LinguaPlone internationalization Product for Plone

LinguaPlone is the standard internationalization product for [Plone](#) that adds language handling support including workflow. It will form the base on which the CMS internationalization capabilities for CORDIS will be based upon.

FS_NUR_LP: LinguaPlone Inclusion in CMS

Include LinguaPlone as the base for the internationalization support in CMS

Deliverables:

- include LinguaPlone product in the release

2.1.12 Enable Email Notification for Workflow Transitions

[Plone](#) has support for automatic email notification upon workflow state transitions. This support is currently not configured and enabled.

Configure and enable email notification inside the [Plone](#) CMS instance so that interested parties can be informed about workflow transitions.

FS_NUR_EN: Email Notification for Workflow Transitions

Configure email notification for CORDIS workflow events

Deliverables:

- working email notification for CORDIS workflow events.

2.2 General Modifications and Extensions

2.2.1 General File Upload instead of only Zip Files

Currently, new content can be uploaded by clicking on the Upload tab and uploading a zip file which is then extracted on the CMS server into the corresponding folder inside the user's workspace. This is cumbersome if the user only wants to upload one file because he has to pack it into a zip file first.

In order to ease upload of new content, the upload tab form will be modified to allow arbitrary file upload. The upload form will be extended by the following two fields:

- a name field for the uploaded object so that the user can specify the name for the new object. This field is not mandatory and can be left empty. See below for an explanation for how the name of the object will be determined.
- a mime type selection field where the user can select the mime type of the object. This field also contains a preselected value automatic which advises the CMS to automatically determine the mime type. See below for an explanation for the mime type detection.

The CMS will create a corresponding object based on the following information:

name (id without extension)

- if the user has entered a name in the name field, that name will be used
- if the user has not entered a name, the name will be derived from the filename of the file that the user uploaded, e.g. directories and the extension will be discarded and the base name be taken as the name.

mime type

- if the user has selected a mime type, his selection will be respected
- if the user did not select a mime type, the mime type will be determined by the `MimetypesRegistry` of the CMS.

extension

The first extension registered for the mime type of the object will be used.

If the user uploads an archive (a file that contains files), e.g. a zip file, the user will have two choices:

- 1 create a file object with the uploaded file as attachment that can be downloaded as is, e.g. handle it as binary content with a suitable mime type.
- 2 extract the files from the archive into the workspace.

The latter choice is the same as what is available in version 3 of the CMS. An intermediate form is presented to the user where he can select the files of the archive that should be extracted.

FS_GME_FU: General File Upload

Support the upload of files inside a workspace folder. Depending on the selected mime type or the extension of the file, the CMS selects a suitable content type and creates an instance named as specified by the user or derived from the path of the uploaded file.

Deliverables:

- support arbitrary file upload inside a workspace folder
- new optional Name field for upload form to control name generation
- if no name specified, the name is derived from the path of the uploaded file
- new Mimetype field for the upload form with a dropdown list of supported mime types
- if no mime type is specified by the user, the mime type is derived from the extension of the path of the uploaded file
- if the uploaded content is an archive, ask the user if he wants to extract or attach to a file
- check permissions for the generated name(s) and inform the user in case of missing permissions

2.2.2 Check uploaded Documents before Object Creation

When importing a document from the workspace into the CMS (pressing the edit tab on a yet unmodified page), the page is checked for markup errors and the user is informed about errors and warnings before the page is imported into the CMS.

The same checks will be done when uploading content via the upload tab, e.g. when the content originates from the users host and not from the server.

When uploading an archive file that the user wants to extract into the workspace (in contrast to a file with attachment), the documents contained in the archive will be checked individually. The status of the check will be shown in the intermediate page that lists the files contained in the archive.

For each contained document, a link can be clicked that shows the check tab for the selected document. The check tab is a page that shows errors and warnings in tabular form with additional details.

FS_GME_CUD: Check Uploaded Documents for Errors

Check documents when uploading them via the Upload tab.

Deliverables:

- uploaded documents are checked for errors
- uploaded documents inside an archive (e.g. zip file) are checked for errors too
- for documents with errors a page with the description of the errors will be shown

2.2.3 Import Time and Modifications after Import

When a document is loaded from the backend (filesystem) or uploaded from the users PC into the CMS, the time when the document was created in the CMS is stored as metadata and can be used by the CMS to detect editing conflicts between different users.

FS_GME_CT: Creation Time for Modified Items

When a document is uploaded or imported into the CMS, the creation time of the CMS object is stored in the CMS and will be used to detect editing conflicts upon delivery.

Deliverables:

- the creation time of objects is stored in the CMS
- upon delivery, the creation time is checked to determine editing conflicts and warn the user.

2.2.4 Move, Rename, Copy, Paste for Workspace

Modifying the name of a file or creating a copy of an existing file is an operation that is currently not directly supported by the existing delivery mechanism (ADS). These operations have therefore not yet been implemented.

These operations will be implemented with the following semantics:

Move, Rename on a name that already exists in the CMS:

If the old name already exists in the CMS (e.g. modified), it will be renamed to the requested new name. If the new name already exists, the user will be informed and required to confirm the operation.

If the new name already exists in the CMS the existing object with the new name will be deleted and the object will take its place.

If the new name already exists on the filesystem, the object will get the new name and upon delivery it will be handled as a modification to the file with the new name.

Move, Rename on a name that only exists on the filesystem:

If the old name only exists on the filesystem, the old name will be marked for deletion.

If the new name exists in the CMS, its content will be replaced by the content of the file with the old name.

if the new name does not exist in the CMS, a new object will be created with the content of the file with the old name.

Copy and Paste:

Copy and Paste will be implemented in the CMS to work as they do for normal CMS ([Plone](#)) content. There will be no difference if the content is on the filesystem or in the CMS.

Copy and Paste will however be restricted to the workspace, e.g. source and destination must be inside the same workspace.

FS_GME_MCP: Move, Copy, Paste for Workspace Content

Implement move, copy and paste for workspace content.

Deliverables:

- Move, copy, paste for workspace content, restricted to the same workspace.

2.3 Multilingual Support

2.3.1 ISO Language Code Extraction from Filename

When importing a document from the filesystem or when uploading a document that contains a two letter language code in its name, just before the extension, and preceded by an underscore, e.g home_en.html, then the language code will be automatically detected as the language of the document with the following consequences:

- the language code will be put into the metadata of the object inside the CMS
- the language code from the filename will be checked against the language attribute inside the document. If they differ, the user will be informed about the fact so that he can select the correct language. The name of the object inside the CMS will be adopted so that it corresponds to the users language selection
- when the document is rendered or delivered, the language attribute will be inserted into the resulting HTML header
- if the language is modified after the document has been entered into the CMS, the document will be automatically renamed to contain the new language code.
-

FS_MLS_RES: Language dependent Versions of Resources

Resources may exist for each of the supported languages in the CMS as well as a language independent version.

2.3.2 Language dependent Versions of Resources

Introduce the notion of a resource that can be available as a language neutral instance and a number of language specific instances (at most 1 per supported language).

FS_MLS_RES: Language dependent Versions of Resources

Resources may exist for each of the supported languages in the CMS as well as a language independent version.

Deliverables:

- support for language dependent versions of a resource

2.3.3 Language dependent Versions and Folder Listing

If a resource is available in different languages, it will be displayed only once in a folder listing with a language indicator (flag icon) for each available language.

FS_MLS_FL: Resources, Languages and Folder Listing

Display only one row per resource in a folder listing with a flag icon for each language that exists for the resource. Clicking on the language icon directs to the language specific version of the resource.

Deliverables:

- in folder listings display flag icons for each language of the resource
- clicking on a flag icon directs to the language specific version of the resource

2.3.4 Resource Language Indicators

For a resource, the current language version is clearly indicated by a flag icon. In addition, other available language version for a resource are shown as language flag icons as well.

The workflow for handling language specific versions in the CMS will be modeled like the language handling in the internationalization products for [Plone](#), e.g. [LinguaPlone](#) or [LinguaFace](#).

FS_MLS_RLI: Resource Language Indicators

Indicate the language of a resource by a flag icon. In addition indicate availability of other language versions of the resource by additional language icons

Deliverables:

- in every CMS template for a resource, display the language of the resource
- in the view template indicate the availability of other languages for the resource by additional language flag icons

2.3.5 Language dependent Versions and Permissions

The permission patterns for users will take into account if the user is allowed to create, edit or deliver a language specific resource.

FS_MLS_RLP: Permission Handling for Language Specific Versions

Permissions are honoured in language specific versions of resources

Deliverables:

- honour permissions in language specific versions of resources

2.3.6 Language Dependent Versions and Composed Documents

Language specific versions of composed documents can be generated by using language specific versions of the components. In this regard, composed documents are not language specific on their own.

The CMS will assist the user in determining the availability of language specific versions. A new tab will be provided that displays the availability of language specific versions of components in tabular form.

FS_MLS_CD: Composed Documents and Languages

Show available languages of the components of a composed document in tabular form in a separate tab

Deliverables:

- create a new tab for displaying component language information
- show available languages for each component inside the tab in tabular form.

2.4 WYSIWYG Display and Editing

2.4.1 WYSIWYG display of (X)HTML documents

Currently the HTML (and CSS and Javascript) of the CMS and a CORDIS page is displayed in one browser window (for normal view). This leads to conflicts or overwriting of styles so the look and feel of both the CMS and the CORDIS page may be mangled to a certain degree. In order to display the two in true WYSIWYG they must be strictly separated.

This can be achieved by displaying the CORDIS page in an iframe inside the CMS navigation. This has unfortunately other side effects which must be handled by the CMS:

- the browser has to load to independent pages which is (a bit) slower than only loading one page
- the iframe is independent, so clicking on a link inside the iframe loads different content into the iframe, but the CMS navigation still remains at the original location, so the user gets easily confused (e.g. clicking on the edit tab may load a different page than the one that is currently displayed inside the iframe)
- the iframe size must be specified (in the CMS page that displays the navigational elements) before the CORDIS page is loaded into the iframe, e.g. the iframe size can only be expressed in units of the CMS navigation page (including relative sizes), so the CMS navigation page cannot be easily resized to the size of the CORDIS page that is loaded into the iframe. This will normally result in an additional scrollbar being displayed for the iframe (as well as a scrollbar for the CMS frame).

The loading of two pages instead of one should be acceptable. Images etc. are also loaded by additional requests to the server, so this should not be a problem.

Clicking on a link inside the iframe can be caught by adding an onclick attribute to all the links in the page, that instructs the parent frame (the frame that contains the CMS navigational elements) to load a different page. This way, the CMS navigation and the iframe with the CORDIS page stay in sync. For this to work correctly, the location that the parent frame is directed at has to be slightly modified so that it loads that page including the CMS navigation. This is however only useful for links that point to locations inside the CMS. For external links, the location will be unmodified but also performed by the parent frame, so the iframe is never directly directed to different location on its own by normal HTML links.

2.4.1.1 Views for (X)HTML documents

The HTML of a document will be displayed independently from the CMS markup so that they do not interfere. So called views are used to display an (X)HTML document in different ways, depending on the context. These views have names and are explained in more detail in the following. They can be accessed directly by appending a / and the view name to the URL of the resource.

Preview

The preview view shows the HTML of the document as it would appear on the CORDIS website, e.g. no CMS navigation is included and the links are untransformed. This view is the standard view when no view is specified in the URL, e.g. navigating to the URL of the document (without any view appended) returns this view too.

View

The view view is the view that shows the CMS navigation with an iframe in which the document's HTML is loaded (in iview view, see below). The CMS navigation provides the usual tabs for editing, uploading, previewing etc, depending on the users permissions.

Iview

The iview view shows the HTML of the document as it would appear on the CORDIS website, but all the links on the page (tags) have onclick attributes that redirect the parent frame (normally the CMS navigation main window for the view view) to a new location where the link is pointing at. If the location is an internal location (e.g. handled by the CMS), then the view view is appended to the URL so that the resulting page is shown under CMS control, otherwise the URL is unmodified and the CMS is left.

This way, a page inside the CMS is always displayed under CMS control if the user does not explicitly request a preview of the page.

FS_WYS_WYG: Iframe based WYSIWYG View for Documents

Display CORDIS webpages inside the CMS navigation as they are displayed on the main website by using iframes.

2.4.2 WYSIWYG Editors and local Link Handling

Every WYSIWYG editor supported by the CMS, e.g. [kupu](#) and [FCKeditor](#), will correctly handle the insertion of local links, independent of the storage location of the local resource, e.g. it does not matter if the resource to be linked to is modified inside the CMS or not (only on the filesystem or later inside ICA2).

FS_WYS_LLH: Local Link Handling in WYSIWYG Editors

Make sure, local links are handled correctly by the WYSIWYG editors

Deliverables:

- correct local link handling by supported WYSIWYG editors

2.4.3 WYSIWYG Editors and Markup Modification

Each WYSIWYG editor has restrictions on the HTML that it can work on. These restrictions will be documented in a content manual. It describes what kind of HTML the WYSIWYG editors accept and what kind of modifications may occur by the editors that are not intended by the users.

FS_WYS_MM: Markup Modifications by WYSIWYG Editors

Describe the markup modifications that WYSIWYG editors perform on HTML content

Deliverables:

- Description of markup modifications performed by the HTML WYSIWYG editors
- Advice for the users, how to avoid certain kinds of markup modifications

2.4.4 WYSIWYG Editing and CSS Styles and Classes

2.4.4.1 WYSIWYG Editor that supports CSS Style Selection

The WYSIWYG editor included in [Plone](#) (on which the CMS is based) only supports a fixed set of CSS styles to be selectable in a toolbar dropdown. In addition, it only has a simple set of toolbar buttons for editing the content in a comfortable way. Therefore a more powerful WYSIWYG editor will be provided in addition to [kupu](#).

There are only two other WYSIWYG editors are supported by addon products in [Plone](#), [FCKeditor](#) and [TinyMCE](#). Although [TinyMCE](#) was used in CMS1 [FCKeditor](#) has been selected for inclusion because it is better integrated into [Plone](#) and easier to customize.

Both editors are comparable on the feature level and appearance.

FS_WYS_FCK: Include FCKeditor as an additional WYSIWYG Editor

Include FCKeditor as an additional WYSIWYG editor in the CMS.

Deliverables:

- Include FCKeditor as WYSIWYG Editor in the CMS

2.4.4.2 CSS Style Selection in FCKeditor

In FCKeditor, the CSS classes that are shown in a dropdown list in the toolbar when the user puts the cursor somewhere inside the edit area is context sensitive to the nearest HTML tag. Which classes to offer for which tag can be controlled by an XML description that FCKeditor downloads from the server.

In order to make the list of offered classes corresponding to the edited document, the document's CSS has to be parsed and a suitable XML representation has to be generated on the fly.

Note!

There is no experience yet with the performance of this approach. If performance is an issue, the CSS parsing results can be cached in a future release.

FS_WYS_CSS: CSS Parsing of Documents

Parse the CSS of each document when it is loaded into the FCKeditor and generate a XML file for FCKeditor's style dropdown.

Deliverables:

- parse the CSS of a document
- generate XML for FCKeditor's style dropdown
- modify the FCKeditor Plone integration so that the dynamically generated XML is loaded

2.4.4.3 FCKeditor Customisation

Note!

This is an additional requirement from Lot 1 that is not contained in the original version of the user requirements document.

The toolbar of the FCKeditor is customizable by someone who knows how FCKeditor is organized and integrated into the CMS, so mostly to developers and perhaps the site administrator.

Inside the CORDIS organization the customization of the editor is however something that should be easy for someone who manages the users and not the site, at least for the aspects of the user interface, e.g. the toolbar.

In order to achieve easier customization of the editor by CMS users without development experience but with the permission to manage site characteristics, a part of the customization possibilities of FCKeditor will be exposed in a template inside the CORDIS site configuration portlet.

FS_WYS_FCKC: FCKeditor Customisation via Portlet

Provide a template inside the CORDIS site customization portlet for the customization of FCKeditor toolbar configuration options.

Deliverables:

- provide a template for the FCKeditor toolbar customization inside the CORDIS site customization portlet.
- describe the customization possibilities in the administration manual.
- Create a new role for the customization of the CMS

2.4.4.4 CSS Dropdown List Customization

Note!

This is an additional requirement from Lot 1 not contained in the original user requirements document.

The CSS classes offered in the FCKeditor dropdown list are all the classes that are allowed for a given tag according to the CSS parsed. This may be too broad for the general users, so the list of CSS should be restrictable to certain elements, explicitly mentioned.

This list could be stored inside the folder so that it applies to all documents therein and the folders, e.g. the CMS searches from the current folder to the root of the workspace.

FS_WYS_CSSC: CSS Customization for Dropdownlist Display

For the CSS class dropdown list XML generation support a filtering mechanism based on a file which lists the allowed classes.

Deliverables:

- when generating the XML for the CSS dropdown list for FCKeditor, take an additional list into account, that contains a list of allowed classes per tag.
- the list of allowed classes can be put into a folder in the workspace with a predefined name. If an object with that name exists, it will be loaded automatically for all documents in the folder.
- a format specification for the list of allowed CSS classes for the HTML tags.

2.4.5 Handle Document Type Setting for HTML pages

Up to version 4.1 CMS2 only supported a fixed document type declaration for HTML pages, independent of the setting in the file when the document was imported or uploaded. This should be changed to allow preserve the document type declaration of a HTML page. As this declaration is outside of the html tags, it must be handled separately to the general HTML handling.

Preserving the setting is important as it determines how browsers render the resulting page. In Addition, a list of allowed document declarations should be provided by the CMT to give the user a useful list of document type declarations in a dropdownlist.

FS_HTM_DT: Preserve Document Type Declaration for HTML

Preserve the document type of a HTML document and provide a dropdown list of supported document type declarations so that the user can select the type easily.

Deliverables:

- provide a dropdown box for selecting a document type declaration from one of the supported ones. CMT provides a list of supported document type declarations for HTML pages.
- preserve the document type declaration of a HTML document if it is error free
- store the document type declaration in the text field of the document.

2.4.6 Active Archive for HTML pages

HTML pages are said to be archived if they contain a corresponding meta tag. As the meta tags are handled dynamically beginning with release 4.2 of the CMS, the user would have to

know the concrete spelling of the corresponding meta tag's name and content attribute. In order to help the user setting a page to archived with a simple mouse click, the meta handling of HTML pages will be enhanced by a checkbox to let the user set a page to archived.

Note!

This change is only a first step in supporting the active archiving of web pages. This method does not allow to set a complete folder (service) to archived. Such a functionality will be added once the content is provided by ICA2 instead of the

FS_HTM_AA: Active Archive for HTML Pages

Provide a checkbox for setting a HTML page to archived, resulting in a meta tag with a predefined name and content attribute to be added to the HTML code.

Deliverables:

- provide a checkbox for setting a HTML page to archived
- store a meta tag with a predefined name and content attribute in the HTML upon save
- the archived flag should not be displayed a second time in the meta tag display data grid and the user should be warned when he tries to add it via the normal meta tag grid.

2.4.7 Search and Replace

Provide a simple Search and Replace facility based on an external search service provided by the CMT. The user can then enter a query to search in all of the web pages and supply a replacement.

The user can then select the pages on which the replacement should be applied, similar to the current folderlisting, e.g. by clicking a checkbox in front of the hit for which he wants the the replace to be performed.

FS_HTM_SR: Search and Replace for HTML Pages

Provide a search interface for doing search and replace on HTML pages.

Deliverables:

- provide a search interface for search and replace where the user can supply a search and a replacement string.
- issue the search query against a service provided by the CMT that returns the paths of the web pages that fulfil the search criteria
- display the search results in a listing with a checkbox in front of each hit.
- provide a Replace button that loads the selected pages into the CMS and applies the replace to it.

2.4.8 Placeholder Insertion Button

Provide a placeholder insertion button on the toolbar of FCKeditor. When the user presses the button, a popup window requests a placeholder name. When the user presses return or a confirmation button (e.g. “create”), the corresponding div tag is inserted at the current cursor position inside the editing iframe.

FS_HTM_PIB: Placeholder Insert Button

Provide a button to insert placeholders.

Deliverables:

- provide a toolbar button in FCKeditor for placeholder creation
- upon button press provide a popup menu with a string field and a confirmation button
- when the user presses the confirmation button or presses “return”, insert a div tag with the placeholder name as id attribute, according to the specification.

2.5 Composed Documents

Note!

The section title has been named Composed Documents although the corresponding section in the user requirements document is called Composed Edit. As some additional requirements have been added in the meantime, the new title better describes the

2.5.1 Visualization of Permissions on Components

When a user edits a composed document in ComposedEdit mode, he can edit the whole page inside the WYSIWYG editor independent of the editing permissions for the individual components. Unfortunately this cannot be avoided technically due to underlying implementation of the editing area inside the browsers (a so called RichEdit widget in IE and Firefox).

In order to not throw away modifications that the user has done to components he does not have permissions for, the user will be warned upon save about the permission problem. He can then choose to either discard the changes to components without permissions (but save the other modifications) or continue editing (to somehow save his modification in a different way, e.g. by copy and past).

FS_CD_CP: Component Permission and Composed Edit

Upon save form ComposedEdit inform the user about modifications inside components for which he does not have enough permissions. Offer options to either save only to components he does have enough permissions for or to continue editing.

Deliverables:

- Upon save after ComposedEdit display an intermediate page that informs about conflicts between permissions and modifications

2.5.2 Visualization of Components as Boxes or Frames

When editing a composed document in ComposedEdit mode, the complete body of the composed document can be edited although the document consists of multiple parts. As such, the user currently has no visual feedback which part of the editor area belongs to which component.

As the CMS has full control over the composed HTML for the edit area, each component could be marked up for the time of editing to easy identifyable without hurting WYSIWYG display to much.

The following possibilities that can be implemented by means of CSS will be implemented and evaluated:

- surround each component by a visible small border
- use a background color

- use a background image

The effects of each will be demonstrated to the CMT and all parties involved will agree on which of the effects to be included in the release.

FS_CD_CV: Component Visualization for Composed Edit

Composed document components will be visualized by CSS during WYSIWYG edit and removed afterwards.

Deliverables:

- implement different CSS visualization methods for composed documents in ComposedEdit
- demonstrate the visual effects
- select visual effects for final release

2.5.3 Checking Component Modifications upon Save

When a user edits a composed document in ComposedEdit mode, the user can modify the content of multiple simple documents at a time (e.g. the template and the content, but the structure may be more fine grained).

When saving the changes, the user should be informed on an intermediate page, which of the simple documents of a composed document that he modified. This intermediate page should also contain a checkbox for each simple document so that the user can select which of the documents should be updated and which modifications should be discarded.

In addition, the intermediate page should provide a button to go back to editing without saving.

FS_CD_CES: Composed Edit Modification Checks

When saving changes to a composed document in ComposedEdit, inform the user about which simple documents have been modified and allow the user to select which simple documents to update.

Deliverables:

- upon save after ComposedEdit display an intermediate page that displays to the user which components he has modified (this will be the same page as the one that informs about permission conflicts)
- for each modified component provide a checkbox so that the user can select which of the components to save
- provide a button to continue editing

2.6 Other Content

2.6.1 Composed Document, XML Component and XSL Stylesheet

A composed document can use a text object with XML content as a component. The XML has however to be transformed to HTML in order to generate the HTML for the composed document.

The transformation from XML to HTML will be described by an XSLT stylesheet. This stylesheet however will not be bound to the XML document itself but to the object that includes it, e.g. the composed document.

This has the advantage that the XML can be rendered differently in different contexts. For composed documents this means that in the placeholder - document grid, an additional stylesheet column will be added that can hold a stylesheet reference.

As the CMS generates previews for composed documents, the XML to HTML transformation must be available to the CMS too. This will be implemented as an external service that will be provided by the CMT and that will receive the XML and the XSL stylesheet and return the resulting HTML for inclusion by the CMS. The API for that service should be simple and network based and will be provided by the CMT.

FS_XML_XSL: Composed Documents, XML and XSL

For a composed document, an XML file can be selected to fill a placeholder. For XML documents as placeholder, an additional XSL stylesheet can be selected in the placeholder - document grid. For preview generation, the XML content and the XSL stylesheet are handed over to a transformation service that is provided by the CMT and that returns the resulting XSL.

Deliverables:

- additional stylesheet column in the placeholder - document grid
- robust API for a XML + XSL to HTML transformation service (by CMT)
- transformation service implementation conforming to the API (by CMT)
- preview generation based on transformation service
- useful error reporting to the user for transformation problems

2.6.2 Dynamic Content, Queries, Search Abstraction Layer

A new text format with a mimetype of text/qry will be supported. It allows the user to specify a query according to the Search Abstraction Layer (SAL) syntax.

These text objects can be used for placeholders in composed documents (where a user can also specify a stylesheet to apply to the result of the query).

When previewed, the query is evaluated by an external server and the result is displayed in human readable form inside the browser.

FS_TXT_QRY: Dynamic Content Items

Support a new mimetype text/qry that when previewed will result in a query evaluation and the result displayed in human readable form inside the browser.

Deliverables:

- install text/qry mimetype inside the CMS and support it for the text/document content type
- make a server address configurable for evaluating a query
- for preview, issue the query to the configured server and return the result to the browser in human readable form.

2.6.3 Coldfusion Templates

Coldfusion templates consist of HTML mixed with tags that contain coldfusion code. In essence, these are text files that can be handled by CMS2. In addition the cfm can be rendered by an external service like for composed documents or dynamic content.

FS_TXT_CFM: Coldfusion Template Support

Support a new mimetype text/cfm or text/x-cfm as the mimetype for coldfusion templates and add a configuration option for a coldfusion server that will render the result for preview.

Deliverables:

- install text/cfm or text/x-cfm mimetype inside the CMS and support it for the text/document content type
- make a server address configurable for evaluating a coldfusion template
- for preview, request a preview from the configured server and return the result to the browser

2.7 Documentation

2.7.1 Online Help / FAQ

Provide the CMS documentation as HTML online versions so that the user interface of the CMS can link to the online documentation for online help.

FS_DOC_OH: Online Help / FAQ

Provide the CMS documentation as online HTML versions to which the CMS user interface can link to provide online help.

Deliverables:

- provide the CMS documentation as HTML

2.7.2 Documentation and Release

The documentation will be provided independently of full software releases so that it can be upgraded independently. Such documentation will contain the documentation in the following formats:

- PDF (Portable Document Format)
- HTML
- source (reStructuredText)

The documentation is written as reStructuredText ([reST](#)) which is normal text with unobtrusive formatting instructions that makes the source version easy to read and maintain. From this, HTML, PDF and other formats (e.g. OpenDocument format) can be automatically generated.

As the sources are maintained in CVS (a revision control system), modifications done by the CMT or Lot 1 can be easily sent back to Lot 2 and incorporated into an update.

FS_DOC_PKG: Documentation and Release

Provide the documentation as an independent package that can be upgraded separately. The package includes the source, PDF and HTML versions.

Deliverables:

- documentation package with user manual, administration manual, content manual, including source, PDF and HTML version

2.7.3 Content-Manual

A lot of questions and reports from CMS users are related to the content and not the functionality provided by the CMS. CMS tries its best to present the content (files, pages, images) in reasonable ways but if the content contains errors the CMS can not always generate a description of the error that is useful to the user. E.g. when the user forgets the "=" sign between an attribute and its value, the CMS is not able to detect that error in a way that it can be reported with a reasonable error message directing to the right location.

The content manual shall explain content related issues when using the CMS such as the markup used for implementing composition or typical errors in HTML and the message that CMS displays in order to give advice to the user on how to fix the problem.

The following sections give an overview of the sections that should be handled in the content manual.

2.7.3.1 WYSIWYG Editors and HTML Markup

The CMS support two WYSIWYG editors in version 4, [kupu](#) and [FCKeditor](#). Each of them has its advantages and disadvantages. They have in common that there are several kinds of markup that they do not support, be it specific tags or attributes for tags.

Especially interactive behaviour such as mouseover etc are problematic because they interfere with the editing task.

Therefore special care must be taken when using these kinds of markup.

FS_DOC_CM1: Content Manual - kupu and FCKeditor

Describe the HTML markup that is supported by kupu and FCKeditor and the HTML markup that is not supported and thus discarded.

2.7.3.2 ComposedEdit and HTML

ComposedEdit displays the composed HTML in a WYSIWYG editor so that the user can modify the content of a page in a WYSIWYG manner even if he does not have the permission to edit the template.

When the user saves the page, the HTML is split up again and only the modified parts are stored back to the CMS.

This is a useful feature but requires some transformations of the HTML during the trip from the CMS to the WYSIWYG editor back to the CMS which modifies the HTML in a way that it differs in whitespace usage (including linebreaks) and replacement of empty tags by the short notation inside one tag with the closing / included.

This has the drawback that the HTML code may look totally different after the user has only done a small modification.

This cannot easily be avoided without major effort. The HTML should render the same though.

FS_DOC_CM2: Content Manual - Composed Edit and HTML

Describe how ComposedEdit modifies the HTML to a certain degree without changing rendering characteristics.

2.8 Versioning

2.8.1 Central Workspace and Backup Copies of Resources

A central workspace will contain all versions that have been delivered for a resource. A naming scheme that contains the date and time of delivery will be used. each resource will be extended by `_YYMMDD_hhmmss`.

The users workspaces will be extended so that they list former versions of resources in a special tab.

FS_VER_BUP: Central Workspace and Backup Copies of Resources

Provide a central workspace with backup copies of former versions of resources. Resource names are extended by the date and time they were delivered. Extend the resource display to offer a tab to display the list of available former versions.

Deliverables:

- provide access to a workspace that contains former versions of resources
- add a tab to the resources that displays available former versions

2.9 Users, Permission Handling and Authentication

2.9.1 CMSPowerUser Role

Add a new CMSPowerUser role that allows to delegate rights to other users. This role will be the default for new users.

FS_UPA_POW: CMSPowerUser Role

Add a new CMSPowerUser role and make it the default for new users.

Deliverables:

- new CMSPowerUser role with permission to delegate its own permissions to other users
- CMSPowerUser role will be the default for new users

2.9.2 User Export and Import

Provide an export and import functionality for registered users, including their modifications inside the workspace.

FS_UPA_UEI: User Export and Import

Provide import and export of users, including their modifications.

Deliverables:

- tab in the CORDIS site configuration portlet to import and export users.

3 Traceability Matrix

The traceability matrix maps user requirement issues to functional specification issues.

UR	Description	FSD Reference
UR-Gen-1	General File Upload via Upload Tab	FS-GME-FU
UR-Gen-2	Mimetype, Resources and Extensions	FS-GME-CUD
UR-Gen-3	CMS Load Time for each Resource	FS-GME-CT
UR-Gen-4	Move, Rename, Copy, Paste for Resources	FS-GME-MCP
UR-I18N-1	ISO Language Code, Filename and URI	FS-MLS-ILC
UR-I18N-2	Language dependent Resources	FS-MLS-RES
UR-I18N-3	Resources, Languages and Folderlisting	FS-MLS-FL
UR-I18N-4	Resource Language Indicators	FS-MLS-RLI
UR-I18N-5	Resource Language and Permissions	FS-MLS-RLP
UR-I18N-6	Composed Documents and Languages	FS-MLS-CD
UR-WYS-1	Separated Display inside Iframes	FS-WYS-WYG
UR-WYS-2	WYSIWYG Editors and Link Handling	FS-WYS-LLH
UR-WYS-3	WYSIWYG Editors and Markup Modification	FS-WYS-MM
UR-WYS-4	CSS Classes and WYSIWYG Editing	FS-WYS-CSSC
UR-WYS-5	Displaying HTML Modifications upon Save	FS-NUR-LXML
UR-WYS-6	Active Archive Checkbox for HTML Pages	FSM-HTM-AA
UR-WYS-7	Search and Replace for HTML Pages	FS-HTM-SR
UR-WYS-8	Preserver Document Type Declaration	FS-HTM-DT
UR-WYS-9	Placeholder Creation Button	FS-HTM-PIB
UR-CE-1	Report Component Permissions in Composed Edit	FS-CD-CP
UR-CE-2	Composed Editing and Component Frames	FS-CD-CV
UR-CE-3	Handle Modifications to Components included twice	FS-CD-CES
UR-CE-4	Displaying Component Modifications upon Save	FS-NUR-LXML
UR-XML-1	XSLT Stylesheet for XML Components	FS-XML-XSL
UR-DOC-1	Online Help / FAQ	FS-DOC-OH
UR-DOC-2	Documentation as independent Package	FS-DOC-PKG

UR	Description	FSD Reference
UR-VER-1	Central Workspace and former Resource Versions	FS-VER-BUP
UR-AA-1	Active Archive GUI	FS-HTM-AA
UR-USER-1	CMSPowerUser	FS-UPA-POW
UR-USER-2	User Registration Export	FS-UPA-UEI, FS-NUR-PAS
UR-QRY-1	Dynamic Content Items	FS-TXT-QRY
UR-CFM-1	Coldfusion Template Support	FS-TXT-CFM