Open source CMS to maintain non-proprietary contents for an information portal

Sampurna, I.

Award date:
2005

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science
Business Information Systems Program

MASTER THESIS

Open Source CMS to Maintain Non-Proprietary Contents for an Information Portal

by
Irwan Sampurna

Supervisors:

Dr. L.M. Aroyo (TUE)
Ir. M. Mooren (C-CONTENT)

Eindhoven, September 2005

Abstract

An information portal is the entry point to access information in the Internet. To attract users, it should have rich and dynamic contents. One strategy is to present non-proprietary contents from multiple sources, while providing contents enrichment and the convenient way for the users to locate a specific content by powerful search functions and content categorization.

To manage and enrich the contents, the implementation of a Content Management System (CMS) is necessary. To support the CMS, there are components to harvest the contents from the sources and to convert them into a uniform XML format. The metadata (information about the contents) are constructed as XML elements within the XML files.

The CMS has an import module to populate the CMS in a batch process. The import module reads the XML files and retrieves the metadata from the XML elements. These metadata are stored in the CMS database, which is RDBMS storage. The CMS users can update the metadata using the CMS user interface.

To enrich the contents collection, there is a link recognizer component that identifies the possible inter-content links based on contextual thesaurus. The links are stored in the CMS database.
Another enrichment process is to present the contents in a hierarchical structure, which is implemented as the table of contents (TOC) constructed by the CMS users.

As an interface between the CMS with the information portal, there is an indexing engine that builds the index set of the contents collection. The CMS has an export module to prepare the contents collection for the indexing engine. The export module merges the metadata and the inter-content links from the CMS database into the XML files. The export module also converts the TOC into a XML file with link attributes to launch the appropriate XML contents.

The CMS is selected from available open source CMS applications. The user interface components are preserved from the selected application, and the import and export modules are developed based on the application's framework and architecture. The application is based on the PHP programming language and MySQL database.

The CMS has been successfully implemented for the information portal about law and legislation in The Netherlands. The CMS contains about 240,000 contents and more than 3,000 TOC entries.

Acknowledgements

I would like to express my gratitude to Lora Aroyo and Michel Mooren as my supervisors as well as tutors during this project. The discussion sessions were very useful for the project and also for this final report.

Furthermore I want to thank people in C-CONTENT for their support during the project. Without their efforts we would not managed to have the Information Portal up and running.

I dedicate this work to my wife for all her love and support.

TABLE OF CONTENTS

1. Introduction and Project Description

1.1. Motivation

Since several years ago the Internet is capable to deliver information in big volumes and with rich presentation format. This situation yields increasing contents delivered via Internet to the users' web browser. From the point of view of the content providers, it is necessary to have a system to manage the contents. This consideration triggered the concept of content management system (CMS).

Based on Robertson in [ROB, 2003], the definition of the CMS is a system that supports the creation, management, distribution, and discovery of information. Robertson stressed out that the management covers the complete content life cycle, from creation to archival. The CMS also provides the ability to manage the structure, the appearance, and the content navigation.
Since we deal (mostly) with web content, the concept of the CMS also mentioned as web management system (WMS), as stated by Robertson. Vidgen et al. in [VID, 2001] use the similar definition on the concept of web content management.

One way to publish the contents in the Internet is using the web portal. Portal means the webpage that provides an entry point to access information in the Internet or an intranet, as stated by Priebe and Pernul in [PRI, 2003]. Because the portal has emphasis on information, it is also defined as an information portal, as mentioned by Mack et al. in [MAC, 2001]. Priebe and Pernul mentioned that the goal of the information portal is to provide the user with a consolidated, personalized user interface to all information that the user needs. Here we combine the statements from both authors to define the information portal as an entry point to access information in the Internet with a consolidated and personalized user interface.

Information about law and legislation is an example of the contents in an information portal. Currently that information is publicly available in several different web sites. The main motivation of the project is to design and build a complete Portal Content Provider system (i.e. system that provides and prepares contents for the information portal) that supports an information portal about law and legislation. The Portal Content Provider system also contains a CMS component to manage the contents.

Unlike the standard CMS, in this project we do not need to have the content creation functionality in the CMS because we use the available law and legislation information. In other words we deal with non-proprietary contents. Relevant with this fact we need to have other components in the system besides the CMS and the information portal, such as a content harvester and a format converter.

One important aspect of using non-proprietary contents for the information portal is we need to have a content enrichment process, for example inter-contents referential links generation. For this purpose we use the metadata (data about the data) to store the additional information about the contents. These metadata are maintained in the CMS.

For the contents storage in the CMS we use RDBMS technology because in the CMS we only deal with structured data. On the other hand, we use XML technology to transfer the contents and the metadata between components in the system until they are published in the information portal.

Another aspect that motivates the project is the consideration to use open source solutions. Due to the open source trend, nowadays there are numbers of open source CMS applications available. One early activity in the project is to analyze several open source CMS packages and select one that is most suitable for the project. The reason of using the open source application is that we do not have to develop everything from scratch, but we still can modify the application to suit the requirements.

1.2. Research Questions

As the foundation of this report and the project, we formulate the following research question:

How can we use open source CMS to maintain non-proprietary contents for an information portal?

The question above can be elaborated into two sub questions that related to the functionality of the CMS and the integration of the CMS with other components that exist in the Portal Content Provider system.

The two sub questions are:

- How can we use combination of XML and RDBMS technologies to help the users enriching the contents, in this case adding metadata information and constructing inter-contents referential links?
- How can we build the import and export components as the interface between the CMS and other components in the system, which can ensure the consistency and integrity of the content flow?

1.3. Goals and Deliverables

Relevant with the explanation in the previous sections, the main goal of the project is to have the complete development life cycle for the CMS. This includes the requirement gathering, open source CMS selection process, design and development of the CMS, contents population, user testing, and user guide documentation.

The deliverables in the project are:

- The requirement specification document of the CMS.
- Analysis and decision on the open source CMS selection process.
- The design and architecture document related to the CMS.
- The implementation of the CMS as working application, including the contents population.

- The CMS user guide documentation.
- The final report for the Master thesis.
- Mid-project presentation.
- Final presentation in the end of the project.

1.4. Project Overview

The project was conducted in the C-CONTENT B.V., The Netherlands. The project team consisted of two project directors, three legal experts who identified and selected the content source, three software developers who designed and developed the CMS and the portal, and several technical advisors from the company.

The phases of the project in chronological order, which also relates to the outline of this report, are:

- The scoping phase, where we described the scope and expectation of the project.
- The requirement phase, in which we described the requirement of the CMS. This part will be explained in Chapter 2.1 of this report.
- The selection phase to analyze the available open source CMS and made decision on one CMS to be used in the project. Chapter 2.2 contains the analysis of the selection process.
- The design phase, where we defined the general architecture of the Portal Content Provider system, analyzed the architecture and design of the selected CMS, and created the design of the additional functionalities in the CMS. This will be explained in Chapter 3.
- The implementation phase, where the real development process was conducted. The implementation included the contents population in the CMS. Chapter 4 contains the documentation about the objects that are developed during this phase.

2. Analysis

After Chapter 1 described the background information of the project, Chapter 2 explains the analysis phase of the project. The analysis process contains two main topics: the requirement analysis and the CMS application selection process.

Before discussing the requirements, first we will look briefly at the domain analysis of the CMS. The domain analysis approach here is based on Kang et al. in [KAN, 1990].
The context of the CMS is about content life cycle management, which includes content creation, review, publishing, and archiving. The input of the CMS can be the content authoring process or content import process. The output of the CMS is the published content that can be presented in electronic format or printed document.

The common features in the CMS are:

- Content authoring tool, which usually is the WYSIWYG (What You See Is What You Get) web-based or plain text editor.
- Content grouping that in standard CMS is defined as content category or presentation menu.
- Content publication styling, for example to adjust the text layout and color setting.
- Content archiving or deletion mechanism. In the CMS the content can be either deactivated (stored but not published) or permanently deleted.

Most of the common properties of the CMS mentioned above were used as the guideline of the analysis process in the project. An exception is regarding the authoring tool, since the project deals with non-proprietary contents as described in Chapter 1.

2.1. Requirements Specification of the CMS

The requirements that described here elaborate the research questions defined in Chapter 1 and they were used as the parameters in the CMS application selection process. Furthermore, these requirements were used as the basis of the design and implementation phases.

The requirement elicitation process was done by interviewing some persons in the C-CONTENT. They are categorized into functional and non-functional requirements. This way of categorizing is based on the approach mentioned by Wiegers in [WIE, 2003].

2.1.1. Functional Requirements

The functional requirements define an explicit scope of the CMS functionalities in the Portal Content Provider system. They describe the processes, the data types, and the required interfaces in the CMS.

The functional requirements of the CMS are:

- The CMS should have facility to construct multiple hierarchical structures that serve as the table of contents (TOC). The TOC will be used as the document hierarchy in the portal and it has different structure for different user group. The nodes in the TOC can be a static expanded title or a link to a document.
- The CMS should have facility to add and update metadata information of the contents. The metadata will be used to store the properties of the contents (i.e. title, document source, publication date, etc.) and the inter-contents links. They will be used to support the functionalities of the information portal, for example the search keywords and the presented document title.
- The CMS should have content categorization structure as a content grouping mechanism. Besides the link to the TOC nodes, the contents shall be grouped by a user-defined category structure, for example by published date.
- The CMS should be able to store the contents in the file system and the metadata information in the database (RDBMS) of the CMS. As described in Chapter 1, the contents are read-only so they can be stored as file system without any necessity of update mechanism. On the other hand, the metadata are updatable so they need to be stored in the CMS database to enable user access.
- The CMS should have content import mechanism to populate the contents. The import mechanism acts as the interface between the CMS and the components 'before' the CMS with respect to the content flow in the system. Detail explanation about the architecture of the system will be discussed in Chapter 3.
- The CMS should have export mechanism to prepare the contents and the TOC for the portal. The export mechanism acts as the interface between the CMS and the portal.
- The CMS should have a version control mechanism. The version control will be used to identify the new or updated contents in the import and export processes, so these processes can be executed incrementally to reduce the execution time.

2.1.2. Non-functional Requirements and Constraints

The non-functional requirements define the quality attributes of the CMS. These quality attributes were used as the technical parameters during the CMS application selection process. In the following section we will see that some non-functional requirements become the important factors during the selection process. Wiegers in [WIE, 2003] also stated that meeting the non-functional requirements often is more important than meeting the functional requirements.

The non-functional requirements for the CMS can be divided into several categories. They are:

Capacity

- The CMS should be able to contain at least 100,000 documents. This number is the approximated amount of publicly available law and legislation documents in The Netherlands at the moment this requirement specification was defined.
- The CMS should be able to contain at least 10,000 TOC nodes. This number is the predicted amount of the TOC structure to categorize the law and legislation documents.

Concurrent Processing

- The CMS should be available during the normal working hours on weekdays, at least for 3 users in the local network of the company. The CMS is planned to be used internally by the company.
- The CMS should also be available during daytimes and nighttimes for the background processing, such as the import and export processes.

Interface

- The CMS should have web-based user interface. The reason is to minimize the hardware dependency due to additional software installation in the users' PC.

Usability

- The CMS should be accompanied by the user guide documents that explain all the available functionalities of the CMS. The user guide documents should also be useful for the user training material.
- The CMS should enable the users to learn and use it properly in no more than 1 day training. This learning period should cover all the functionalities of the CMS.

As an addition to the non-functional requirements, there are some constraints related to the technology:

- The CMS should be selected from available open source CMS applications. This is relevant with the motivation described in Chapter 1.
- The CMS should use RDBMS as the storage system and also have capability to handle XML files. The RDBMS storage system will be used to store the structured information of the contents and the metadata. The XML files will be used to transfer the contents and the metadata between the CMS and other components in the system. By using XML files we can have flexible metadata definition for different content types.
- The CMS should support standard character encodings such as ISO 8859 and Unicode standards. This is necessary because the contents are mostly in Dutch language and contain special European characters.
- The CMS should use MS Windows as the operating system. The reason is to enable the future integration between the CMS and the indexing engine developed by the C-CONTENT, which is based on Microsoft technology.

2.2. CMS Application Selection

The requirements explained in the previous section are used as the basis of the CMS application selection process. The selection process can be elaborated in the following activities:

- **Research on Available Open Source CMS**

  Open source community web sites for CMS applications were inspected and analyzed. The information is compared with the requirement specifications, both the functional and non-functional aspects. The most useful sources during the selection process are SourceForge.net (http://sourceforge.net) and opensourceCMS (http://www.opensourcecms.com).

  Besides the relevancies with the requirements, the selection process also considers the facts about the CMS applications. These facts are:
  - Number of live systems that use the CMS applications.
  - Size of the developers or users community.
  - The age of the CMS applications.
  - The user feedbacks or comments about the CMS applications.
  - The rate of responsiveness of the developers regarding bugs and difficulties.

  The research step gave 19 candidates of CMS applications. Eight from them are proprietary applications with commercial license, so the number of candidates was reduced to 11. The majority of the candidates use the combination of Apache, PHP, and MySQL technologies, and they are compatible with multiple operating systems.

- **Analysis of Standard CMS Application Functionalities**

  In this step, the CMS application candidates were installed and their functionalities and performance were analyzed.
  During this step the number of candidates was again reduced. The elimination factors in this step are:
  - Mismatch of the provided functionalities with the requirements. The functional requirements were the main focus, but it was also possible to analyze the usability while checking the provided functionalities of the CMS.
  - Problem in performance in terms of the standard functionalities of the CMS application.

  The following table contains the summary of the analysis process on the 11 CMS candidates.

CMS Candidates	Installation	TOC Hierarchy	Content Category	File and RDBMS Storage	Default Performance
Apache Lenya	√	×	×	×	×
Ariadne	√	×	√	×	√
Exponent	√	√	√	√	√
eZ publish	√	√	√	×	×
LucidCMS	√	×	×	×	√
Mambo	√	×	√	×	√
MD-Pro	√	×	×	×	√
OpenCMS	√	×	×	×	×
OpenPHPNuke	√	×	×	×	√
PostNuke	√	√	√	√	√
Silva	√	×	×	×	×

The result of this analysis step was the best two CMS candidates, as highlighted in the table.

- **Final CMS Selection**

  The two selected CMS applications that passed the previous analysis activities are Exponent CMS and PostNuke CMS (with ContentExpress module). The table below indicates the matching of the two CMS with the functional requirements.

Functional Requirement	Exponent	PostNuke
Facility to construct multiple hierarchical structures that serve as the TOC hierarchy	Yes	Yes
Facility to add and update metadata information of the contents	With modification	With modification
Content categorization structure as a content grouping mechanism	Yes	Yes
Ability to store the contents in the file system and the metadata information in the database (RDBMS) of the CMS	Yes	Yes
Content import mechanism to populate the contents	No	No
Export mechanism to prepare the contents and the TOC for the portal	No	No
Version control mechanism	Simple	Simple

Note that both CMS do not match the requirements related to the import and export mechanisms. This is general situation for all the analyzed CMS, so principally these two requirements need to be developed as additional functionalities.

The following tables show the matching of the two CMS with the non-functional requirements.

Non-Functional Requirement: Capacity	Exponent	PostNuke
Ability to contain at least 100,000 documents	**Performance problem**	Yes
Ability to contain at least 10,000 TOC nodes	**Performance problem**	Yes

Non-Functional Requirement: Concurrency	Exponent	PostNuke
Available during the normal working hours on weekdays, at least for 3 users	Yes	Yes
Available during daytimes and nighttimes for background processing	Yes	Yes

Non-Functional Requirement: Interface	Exponent	PostNuke
Have web-based user interface	Yes	Yes

Non-Functional Requirement: Usability	Exponent	PostNuke
Accompanied by the user guide documents	With addition	With addition
Maximum one day training	Yes	Yes

Constraint: Technology	Exponent	PostNuke
Open source CMS application	Yes	Yes
Use RDBMS as the storage system and also have capability to handle XML files	Yes	Yes
Support standard character encodings	Yes	Yes
Use MS Windows as the operating system	Yes	Yes

As we can see from the tables, the key factor in the final selection process was the performance difference between Exponent CMS and PostNuke CMS. During the performance testing, the Exponent CMS was tested with 10,000 TOC nodes and it gave the response time of between 60 and 90 seconds per navigation click. On the other hand, testing with 30,000 TOC nodes in the PostNuke CMS gave response time in average of 8 seconds per navigation click.

Based on the significant performance difference between the two CMS, the final selected CMS was the PostNuke CMS.

3. Architecture and Design

This chapter describes the architecture of the Portal Content Provider system as well as the CMS. The Portal Content Provider system is the integrated components that prepare the contents for the Information Portal.
The approach of this chapter is first presenting the architecture of the complete Portal Content Provider system including some explanations about the components in it. This broad overview can help us understanding the interaction between the CMS and other components and the importance of the CMS in the system.
The second part describes the architecture of the CMS itself, which is divided into several functional modules.

Besides the architecture, this chapter describes the design of the CMS database and site navigation. These designs are the underlying model for the implementation phase to develop the working CMS.

3.1. General Architecture

The architecture of the Portal Content Provider system can be found in Figure 3.1.

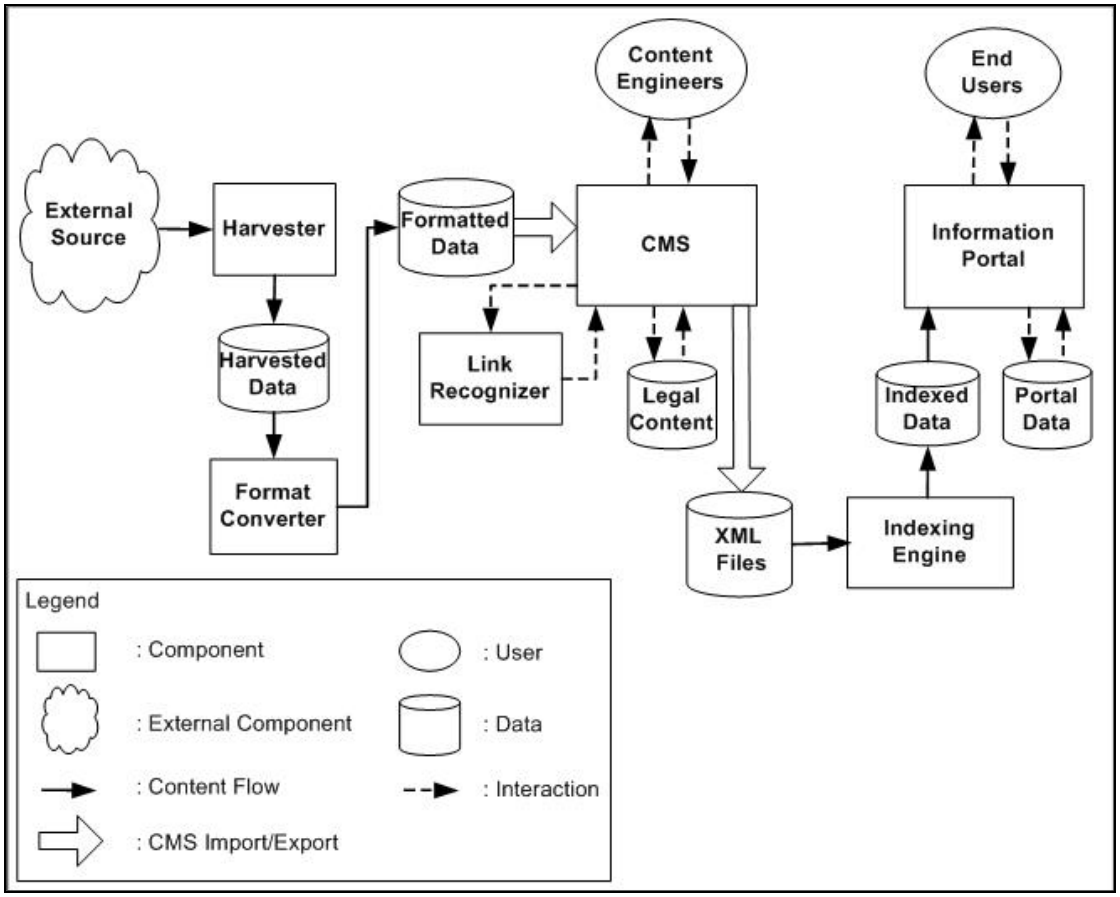


Figure 3.1 Architecture of the Portal Content Provider System

The explanation of each component in the system can be found below.

Harvester

The harvester is the component that retrieves the content from the external sources. It should maintain the list of retrieved content and should be able to detect the new and updated content.

Format Converter

This is the component that formats the content as XML files and identifies the metadata information from the contents and set them as XML elements in the header part of the XML files. This component also maintains the presentation format uniformity of the contents, so that the portal users will see a standard document style.

CMS

This is the component to store the content and to construct the TOC. The CMS also stores the additional metadata information that necessary for the information portal. The CMS has import and export modules as interface components with the other components in the system.

The main users of the CMS are the content engineers who responsible for TOC management and execute import and export processes in the CMS.

Link Recognizer

The link recognizer is the component that analyzes the contextual terms in the contents collection and generates the inter-content links based on those contextual terms. The link recognizer uses the terms reference list, which contains list of law and legislation terminologies and abbreviations, to generate the links.

The result of the link recognizer is a file containing the inter-content links information. This information is imported in the CMS as metadata records for the relevant contents.

Indexing Engine

This is the component that creates the indexes on the contents, to enable the end users to retrieve the specific content by browsing through the TOC or by searching using specific keywords. We use eXtrect®, the XML-based indexing engine product of C-CONTENT. The eXtrect® stores the contents in its own database that will be used by the information portal.

Information Portal

The information portal is the web-based application where the users can view the contents based on specific TOC and to perform content search, either using full text search or using search keywords provided by the indexing engine.

The main users of the information portal are the citizens, the lawyers, the law students, etc. Each of the user groups will have different TOC to view the contents.
Besides the contents, the web portal also provides several additional functionalities such as user forums, news, and polls.

3.2. Architecture of the CMS

The CMS architecture is based on the existing architecture of the PostNuke CMS. The CMS architecture diagram can be found in the Figure 3.2. The TOC Management module is originally provided by the PostNuke CMS. The Import and Export Modules as well as the Links Generation are new modules that were developed in this project.

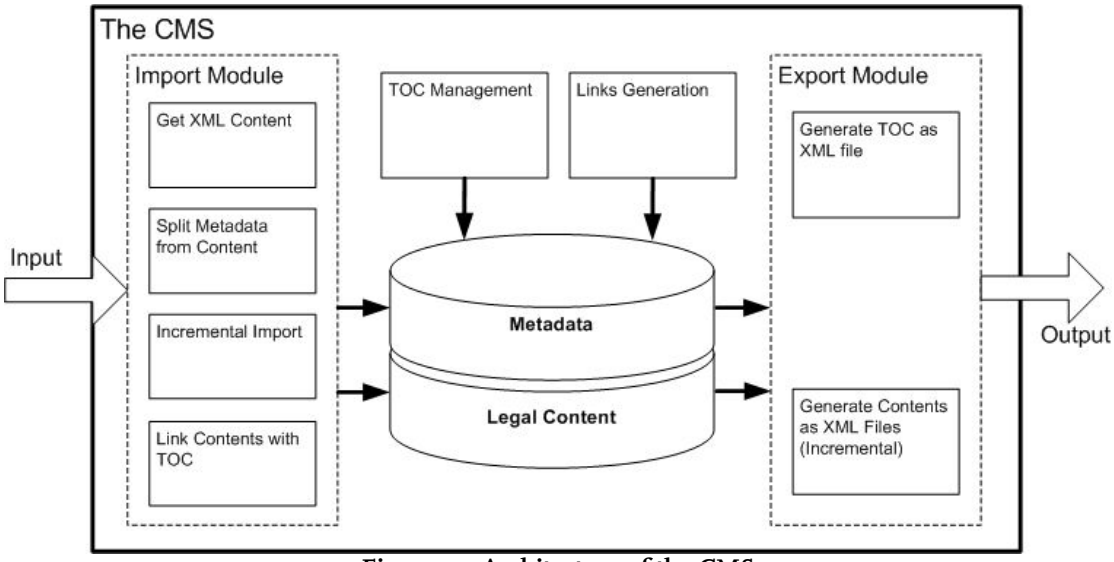


Figure 3.2 Architecture of the CMS

3.2.1. Content Import Module

The import module is the interface component between the CMS and the previous components in the data flow (i.e. the Harvester and the Format Converter).

This module collects the contents from the resulted file structure of the Format Converter and copies them to the file storage of the CMS. The process uses the timestamp of the files to filter the amount of the files that being copied. During the copy activity the import module executes the followings in sequential order:

(1) Copy the documents from the storage of the Format Converter module to the storage of the CMS.
(2) Optional: construct the category hierarchy based on the file structure of the contents. This categorization will be used to group the contents.
(3) Record the documents information in the content table of the CMS.
(4) Record the metadata in the metadata table of the CMS.

(5) Optional: link the contents to the appropriate menu leaves in the TOC in the CMS.

Steps (2) and (5) are optional depend on the content type. This is the decision of the users whether to group the contents using the category structure or using the TOC structure.

3.2.2. TOC Management

The TOC is the document structure that being defined by the content engineers. The content engineers construct the TOC using the menu creation function provided by the CMS. The TOC can have multiple hierarchy levels and unlimited number of menu nodes. For the specific leaves that refer to a document, the content engineers provide the document title as the referential key to enable the import module links the appropriate contents to the leaves of the TOC.

We can have multiple sets of TOC in the CMS to handle multiple user groups with different presentation structure.

3.2.3. Links Generation

This module is used to create inter-content referential links. Currently this module employs the link recognizer component.

The referential links information is stored as the metadata information of the referring documents. This metadata information is used by the export module to generate the complete document structure containing both the referred and the referring documents.

3.2.4. Content Export Module

The export module is the interface component between the CMS and the indexing engine.
This module converts the documents in the CMS into XML files. During this conversion the export module merges the metadata information as additional XML elements in the XML files. The export process is done incrementally based on the flag in the document table that indicates if the document is new or has updated after the previous export execution.

There is also export process to create the XML TOC that will be used by the eXtrect®. The XML TOC is the transformation from the TOC structure in the CMS to XML file. The menu leaves that refer to specific document will be transformed into a XML element with specific XML link to the appropriate XML document. The XML TOC also contains the inter-content referential links information that generated from the metadata information.

3.3. Database Design of the CMS

The entity-relationship (E-R) diagram of the CMS database can be found in the Figure 3.3.

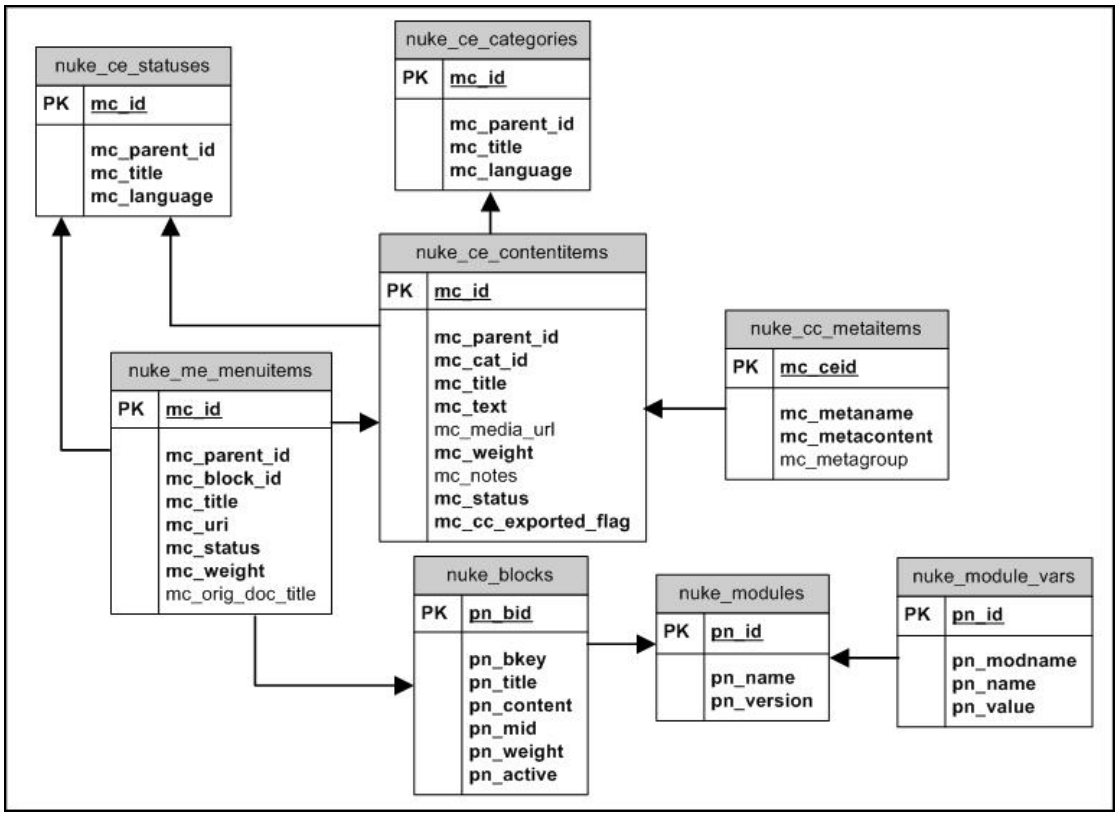


Figure 3.3 CMS Database E-R Diagram

Most of the tables and the relational structures were provided by the PostNuke CMS and the ContentExpress module. The new table is the `nuke_cc_metaitems` table, which contains the metadata information related to specific contents. The original database design of the PostNuke CMS contains no explicit foreign key constraints. All table references are done in the application (i.e. PHP) layer instead in the database layer.

The description of each table is as follows.

- `nuke_ce_statuses`
  This is a reference table containing status of the document. Currently the valid statuses are 'Preview' and 'Posted'. The statuses are referred by the content (`nuke_ce_contentitems`) and the menu (`nuke_me_menuitems`).

- `nuke_ce_categories`
  This table contains content category data. The category can have hierarchical structure by using `mc_parent_id` column as a referential key to the parent category.
  The categories are used to group the content (`nuke_ce_contentitems`).

- `nuke_ce_contentitems`
  This is the table to store the data about the content. This table is populated by the import module. The contents are grouped by category using `mc_cat_id` as referential key to the `nuke_ce_categories` table.
  The `mc_text` column contains the URL to open the content in separate web browser. The `mc_notes` column contains the information of the file location of the content. The `cc_exported_flag` column is used to indicate if there is a new or updated document that needs to be exported into XML file.

- `nuke_cc_metaitems`
  This is the table to store the metadata information for the contents (the metadata element name and the metadata element value). Each record refers to `nuke_ce_contentitems` table using `mc_ceid` column as the referential key.

- `nuke_me_menuitems`
  This table contains the menus of the TOC. The menu hierarchical structure uses the `mc_parent_id` column as the referential key to the parent menu.
  The `mc_title` column contains menu name and the `mc_orig_doc_title` contains the original document title. The original document titles are used during the import process to link the menu to the appropriate content. The link to the content is stored in the `mc_uri` column.

- `nuke_blocks`
  The block is used to group the menu set (the TOC). If we have multiple TOC, then we can create multiple blocks that have different menu hierarchy.

- `nuke_modules`
  This table contains the information about existing module in the PostNuke CMS. With respect to the CMS, we only use the ContentExpress module.

- `nuke_module_vars`
  This table contains global variables that can be used in the CMS. We use this table to store parameter values for the import and the export processes.

3.4. Site Navigation Design of the CMS

The WebML, a graphical and formal design model dedicated for web application, is used as the basis of the design diagram. The WebML was created by researchers in the Politecnico di Milano, Italy. More information about WebML is described by Ceri et al. in [CER, 2002].
The legend for the WebML diagrams in this section can be found in Appendix C.

The CMS contains two important areas related to the functional requirement of the CMS. The high level view of the two areas and the main page of the CMS can be found in the Figure 3.4.

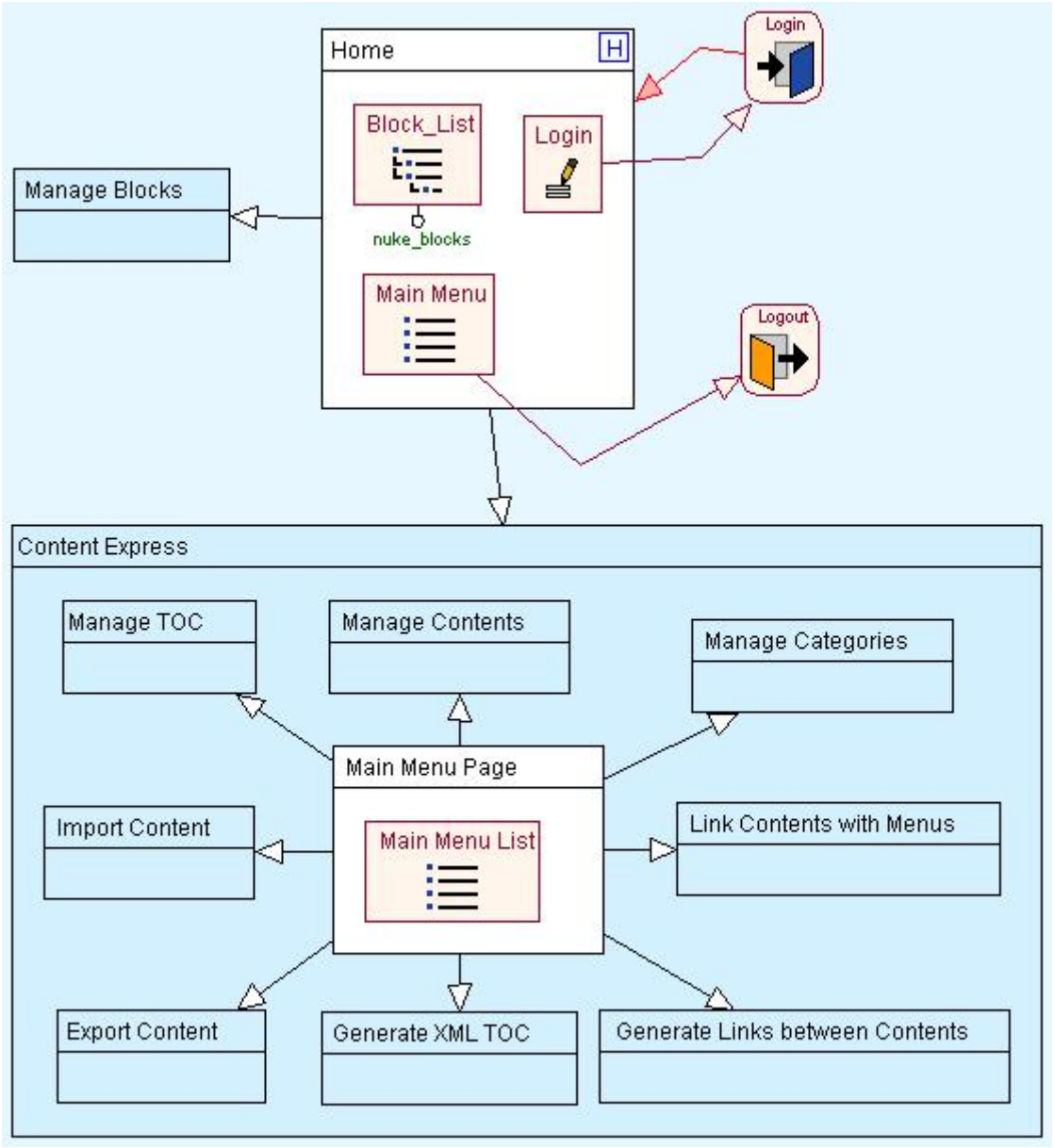


Figure 3.4 Global Site Navigation Design

The 'Manage Blocks' area is related to the menu block management and the 'Content Express' area is related to the content and TOC management.

The detail explanation of each area is as follows.

Home Page
The home page is the main gate to access the functionality of the CMS. The page contains login area, menu block area, and the main menu. The main processes in the home page are the login and logout of the users. There are also navigation links to the Manage Blocks and Content Express areas.

Manage Blocks Area
This area describes the functionality to manage the menu block. The detail view is shown in the Figure 3.5.

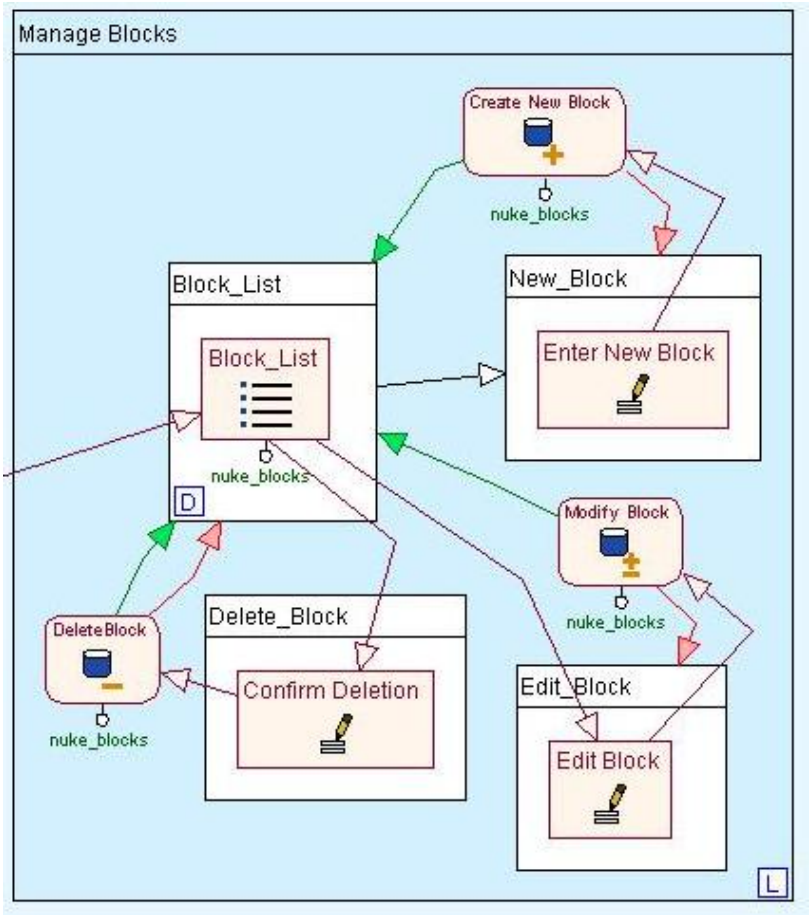


Figure 3.5 Blocks Management

The default page if we access this area is the Block List page, where we view the list of existing menu blocks. Furthermore there are three possible actions that can be performed on the menu blocks:

(1) New Block
(2) Edit Block
(3) Delete Block

All the actions in the Manage Blocks area are related to the `nuke_blocks` entity in the database.

Content Express Area

This area expresses the main content management functionalities in the CMS. The main page in this area contains the list of functionalities in the area. They are:

(1) Manage TOC
(2) Manage Contents
(3) Manage Categories
(4) Import Content
(5) Export Content
(6) Generate XML TOC
(7) Link Contents with Menus
(8) Generate Links between Contents

These functionalities are described as sub areas in the Content Express area.

Manage TOC
This area describes the functionality to manage the TOC or the navigation structure to view the contents. The detail view of this area is as shown in the Figure 3.6.

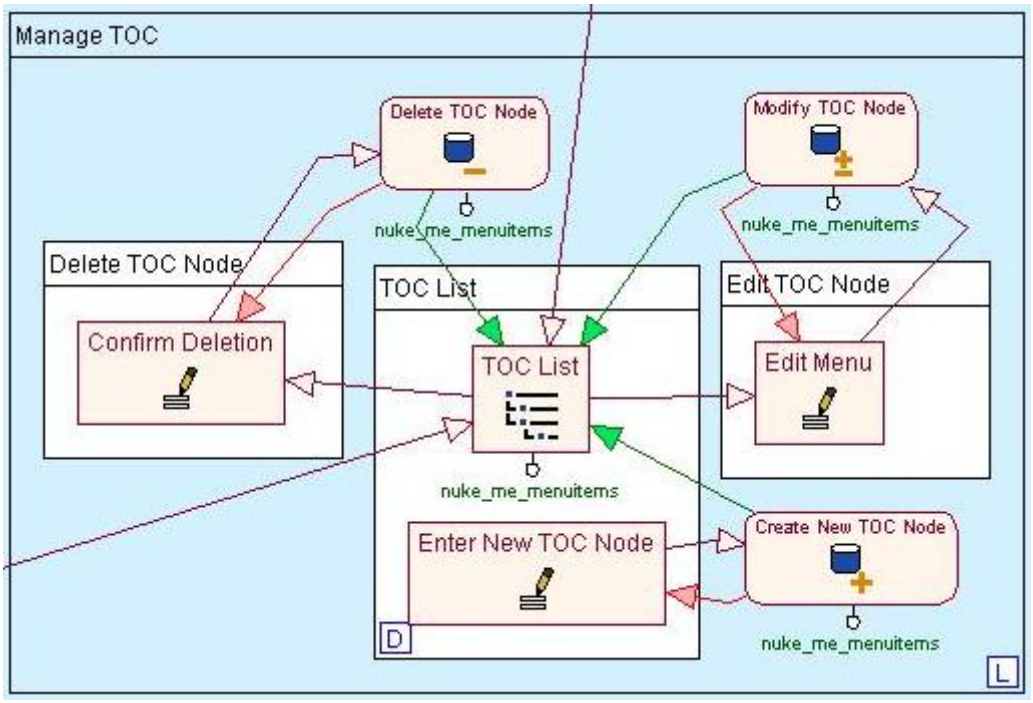


Figure 3.6 TOC Management

The main page of this area is the TOC List page that shows the hierarchy of the TOC, grouped by the blocks. There are three possible actions that related to the menus, which are:

(1) Create New TOC node
(2) Edit TOC node
(3) Delete TOC node

All these actions are related to the `nuke_me_menuitems` database entity.

Manage Contents
This area described the functionality to manage the contents of the CMS. The detail view can be seen in the Figure 3.7.

The main page is the Content List, which displays the contents based on the hierarchical categories.
From the list, there are three possible actions:

(1) View the individual content
(2) Launch the external application to generate links between contents
(3) View the existing links between contents

In the link list page, there is a delete link action.

The contents are stored in the `nuke_ce_contentitems` entity and the links are stored in the `nuke_cc_metaitems` entity.

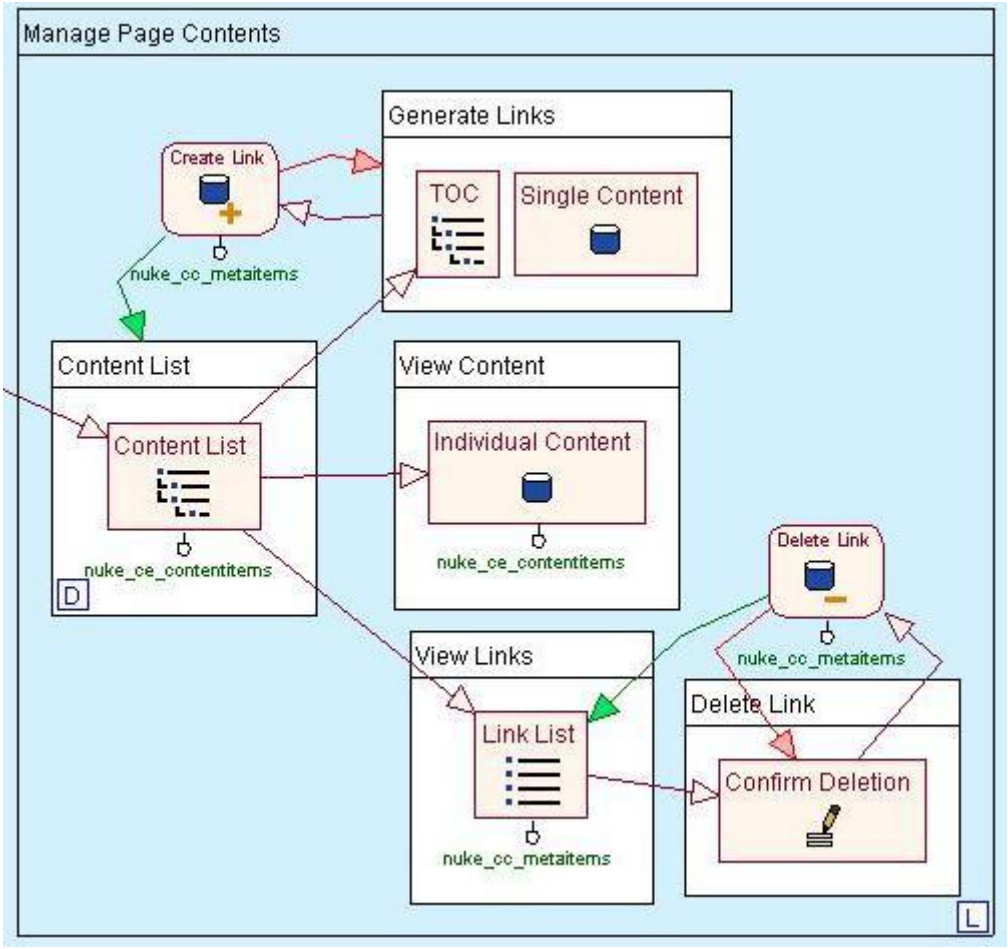


Figure 3.7 Contents Management

Manage Categories

This area defines the functionality to manage the categories that used to group the contents. The detail view of the area is shown in the Figure 3.8.

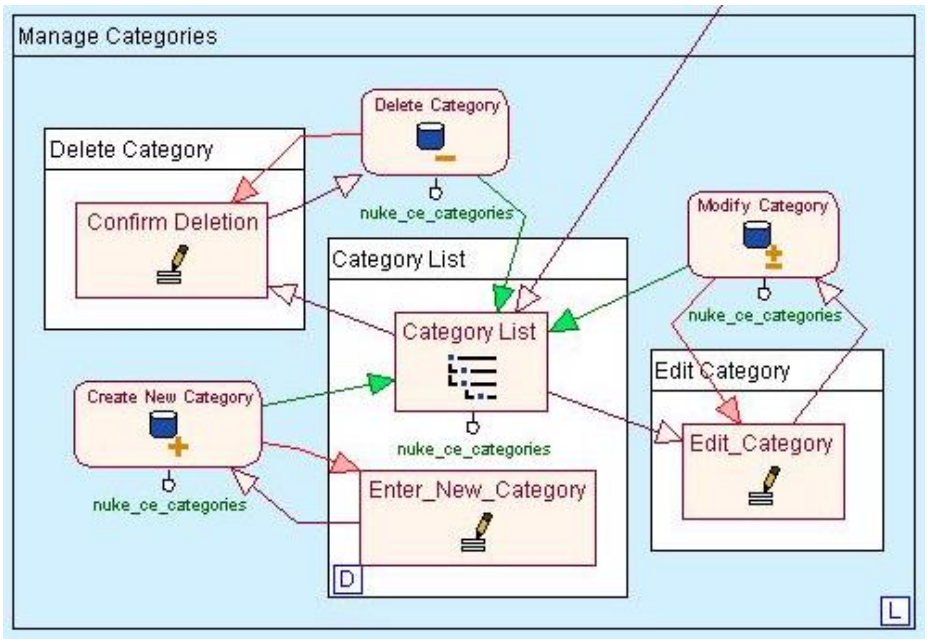


Figure 3.8 Categories Management

The main page is the Category List page that displays the hierarchy list of the categories. There are three possible actions related to the category management:

(1) Create New Category
(2) Edit Category
(3) Delete Category

All the actions are related to the `nuke_ce_categories` database entity.

Import Content

This area describes the content import process of the CMS. It contains the main page that displays the content category selection for the import process and the following page that displays the parameters that used by the import process. The import parameters are retrieved and stored in the `nuke_module_vars` database entity. The import process itself updates or inserts the `nuke_ce_contentitems` database entity.

The detail view is shown in the Figure 3.9.

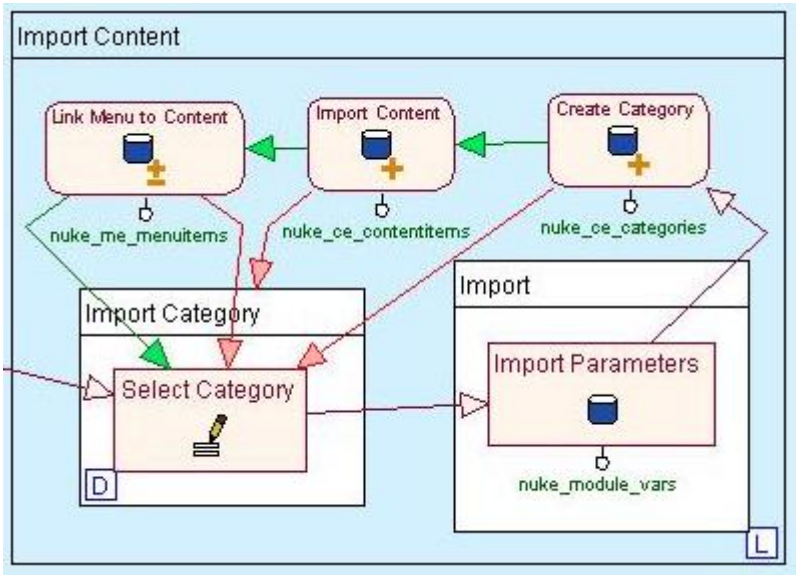


Figure 3.9 Import Content

Export Content

This area defines the content export process in the CMS. It contains a default page that displays the export parameters. The export result is XML files and the users will get success or failure acknowledgement.

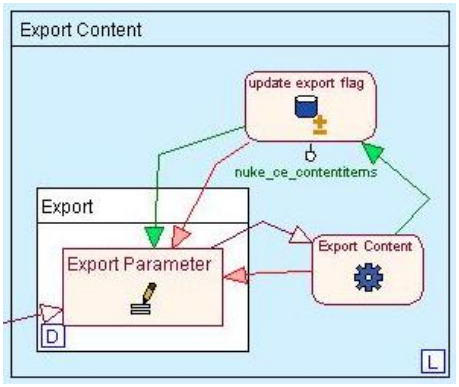


Figure 3.10 Export Content

Generate XML TOC
This area describes the XML TOC generation process. There is a default page that shows the generation parameters. The generation result is a XML file. The users will get notification if the process is successful or fail.

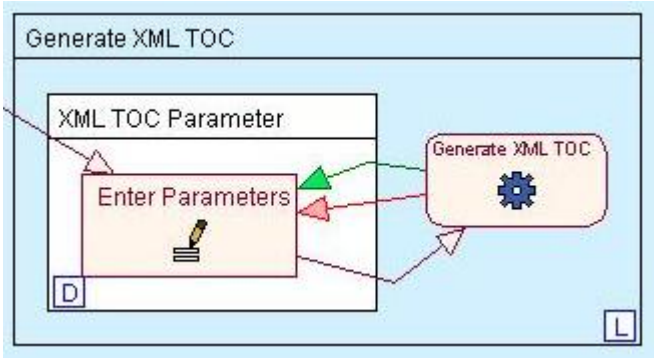


Figure 3.11 XML TOC Generation

Link Contents with Menus
This area describes the process that maps the menus with relevant contents. The mapping is generated based on unique referential links between the menus and the contents. This process updates the link information in the `nuke_me_menuitems` database entity.

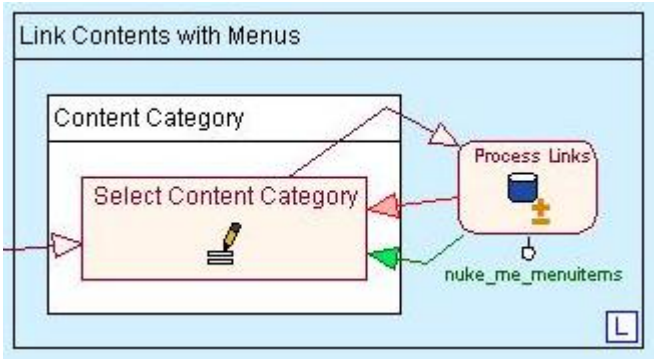


Figure 3.12 Link Contents with Menus

Generate Links between Contents
This area describes the process that updates the inter-content links information. The process reads an input file that contains the links information, and updates the `nuke_cc_metaitems` database entity with the information.

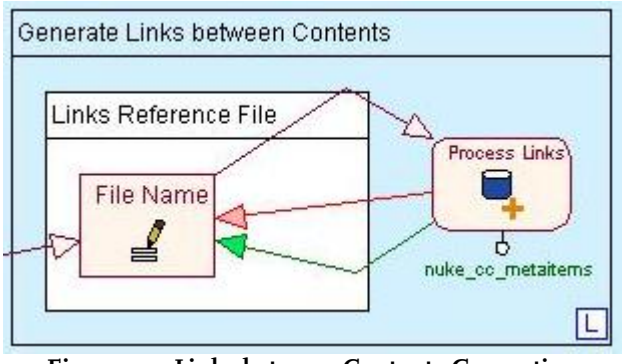


Figure 3.13 Links between Contents Generation

4. Implementation

The implementation of the CMS was done based on the architecture and design that were described in Chapter 3. The import, export, and metadata management components are new and the other components were already provided by the PostNuke CMS, for example the TOC, categories, and contents management. Until some extents these existing components need to be modified to improve the navigation flow, performance, and integration with the new components.

The implementation used the PHP programming language, which is the language used by the PostNuke CMS. The new components were developed as new classes in the existing development framework of the PostNuke CMS, specifically the ContentExpress module. They inherit the global properties and methods provided by the PostNuke CMS and the ContentExpress module, and fully integrated with the existing classes. References about the PHP language were the PHP web site (http://www.php.net) and [ATK, 2003], a PHP programming book by Atkinson and Suraski.

The implementation also covered the data population for the first release of the information portal. We will discuss the data population in the separate section in this chapter.

4.1. Import Module

The import module was developed as one class, which is named as `CCImportXML` class. This class has several functions that construct the pages related with the import module and several logical functions that called during the import process execution.

The important functions in the `CCImportXML` class are:

- `admin_cc_modify_configtype`, `admin_cc_modify_config`, and `admin_cc_update_config`

  These functions construct the import configuration pages that used by the users to specify import parameters. As described in the design in Chapter 3, the import process is filtered by content category selection.
  The `admin_cc_modify_configtype` function constructs the initial page where the users can select the content category that they will configure.
  The `admin_cc_modify_config` function constructs the second page where the users can enter the import parameters. Currently the parameters are:
  - Directory name of the import source files.
  - Target directory name (the CMS file storage).
  - Web alias that used to construct the URL to access the imported files.
  - XML elements name that should be extracted and stored as metadata information in the CMS.

  The `admin_cc_update_config` function handles the verification and database update processes on the parameters that passed by the

admin_cc_modify_config function. These parameters are stored in the nuke_module_vars table.

- admin_cc_enterrunimporttype, admin_cc_enterrunimport, and admin_cc_generaterunimport

  These functions construct the pages where the users execute the import process.
  The admin_cc_enterrunimporttype function constructs the initial page where the users can select the content category that will be imported.
  The admin_cc_enterrunimport function constructs the page that displays the import parameters and the previous import timestamp for the selected category. The users can execute the import process from this page.
  The admin_cc_generaterunimport function handles the execution process and calls the import API function.

- api_cc_import_content

  This is the main API function in the import module. This function handles the whole logic of the import process as described in Section 3.2.1. The parameters of this function are the import parameters, the previous import timestamp, and the content category. The detail execution flow is as follows.

```
1. Verify if all parameters are set.
2. Set 'author' variable depends on the content category.
3. Get the structure of the import source directory.
4. Loop for each directory structure:
    5. Get the timestamp of the directory.
    6. If directory timestamp is greater than the previous import
       timestamp, do the following:
        7.  Copy the directory to the target directory.
        8.  Optional based on the content category: create category
            structure in the CMS database based on the directory
            structure.
        9.  Get the files in the directory. It is filtered based on
            the file extension (.xml files).
        10. Loop for each file:
            11. Get the metadata information from the file.
            12. Record the metadata information in the CMS
                database.
            13. Construct the content title. This can be based on
                the file name, the metadata information, or a
                reference table.
            14. Construct the content unique identifier. This can
                be based on the file name, the directory structure,
                or the combination of them. It is stored in the
                'notes' variable.
            15. Construct URL to view the content as web-based file
                using style sheet constructed with XSL.
            16. Prepare variables to record the content information
                in the database.
            17. Check the content existence in the database, based
                on the combination of 'notes' and 'author'
                variables.
                18. If content exist, then process update.
```

```
                19. Else if content does not exist, then process
                    insert.
            20. Optional based on the content category: create link
                from the content to the appropriate menu nodes in
                the CMS. This is based on the content title or the
                content unique identifier.
        21. End loop for each file.
    22. End timestamp condition block.
23. End loop for each directory.
24. Update the import timestamp.
25. End of the function.
```

Note that in step 18 and 19 the `cc_exported_flag` column in `nuke_ce_contentitems` table is set to 0 (zero), indicating that the content is new or updated and should be processed by the export module.

Following the modularization concept, most of the steps in the API function, such as the database insert or update, are constructed as separate functions in the `CCImportXML` class.

The visual representation of the `CCImportXML` class is as shown in the Figure 4.1. In the Figure 4.1 we can see the relations between the functions in the `CCImportXML` class including the functions that are called by the `api_cc_import_content` function.

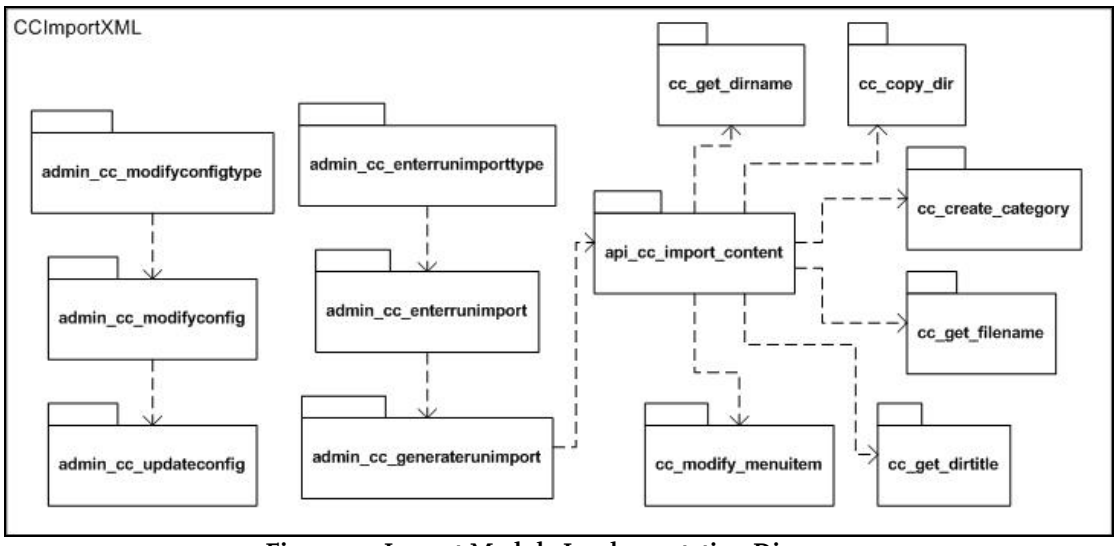


Figure 4.1 Import Module Implementation Diagram

The sample screenshot is shown in Figure 4.2. It shows the import parameter to be confirmed by the user before the import execution.

Figure 4.2 Import Screen

4.2. TOC Management

The sample TOC management screenshot is shown in Figure 4.3. It shows the existing TOC hierarchy in the upper part, and the forms where the user can create a new TOC entry. The complete CMS user guide documentation can be found in Appendix B.

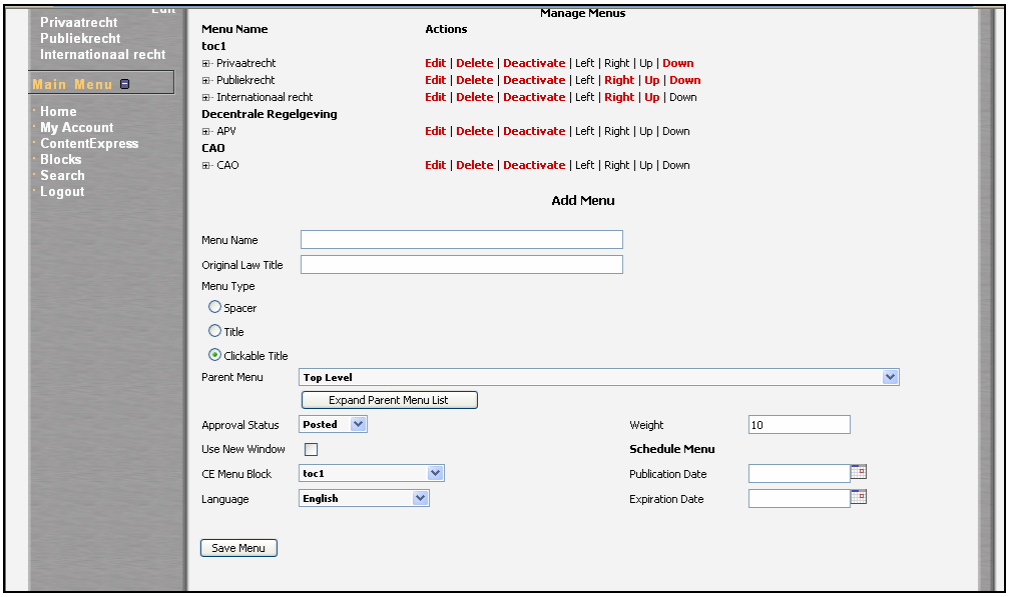


Figure 4.3 TOC Management Screen

The TOC management was implemented using the existing class from the ContentExpress module, which is the `MenuExpress` class. This class contains the functions that provide the menu management feature in the CMS.

The functions in the `MenuExpress` class are:

- The generic functions, which are the `api_get`, `api_getall`, and `admin_build_form` functions.

- To create a new menu, which is covered in `admin_new`, `admin_create`, and `api_create` functions. The `admin_new` function uses the `admin_build_form` function that constructs the form page where the users fill in the menu information, the `admin_create` function handles the data verification and API calling, and the `api_create` is the API function that executes the database insert to the `nuke_me_menuitems` table.

- To modify an existing menu, which is handled by `admin_edit`, `admin_update`, and `api_update` functions. The `admin_edit` function calls the `api_get` function to retrieve the menu information form the database, and calls the `admin_build_form` function to construct the form page where the users can edit the information. The `admin_update` function handles the data verification and API calling. The `api_update` is the API function that processes the database update in the `nuke_me_menuitems` table.

- To delete an existing menu, which is handled by `admin_delete` and `api_delete` functions. The `admin_delete` function calls the `api_get` function to retrieve the menu information form the database, provides deletion confirmation page for the users, and also calls the API function. The `api_delete` is the API function that executes the database delete process in the `nuke_me_menuitems` table.

- To display the menu hierarchy, which is handled by `admin_view` function. This function calls the `api_getall` function to retrieve all the menu records from the database, and constructs the menus based on the hierarchical structure.

The Figure 4.4 presents the visual representation of the `MenuExpress` class.

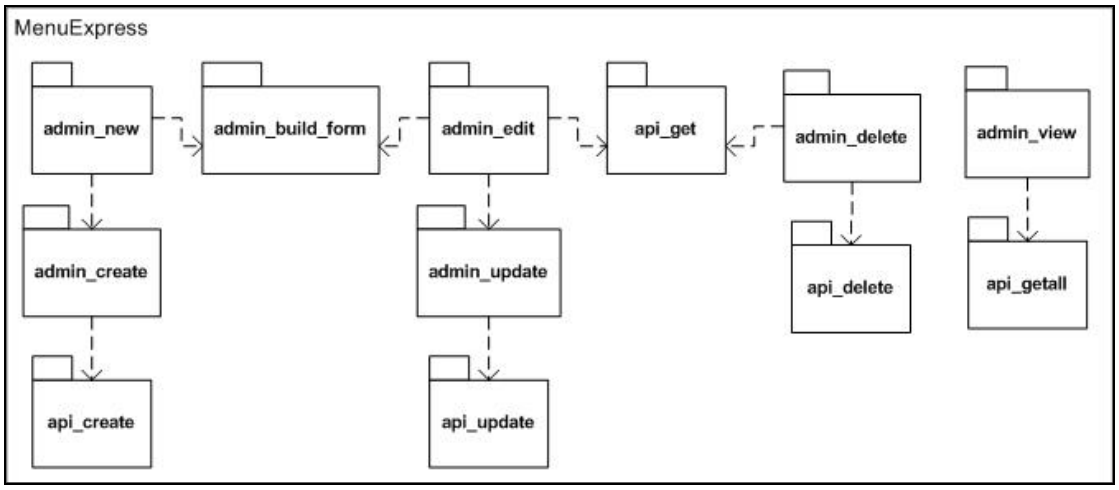


Figure 4.4 TOC Management Implementation Diagram

The menu has several variables, and the most important ones are:

- The menu title that is displayed to the users.
- The original document title, a new variable that is used to link the menu with the content.

- The menu type, either as a static text, a clickable title, or a title with link to the content.
- The parent menu title, which is used as the identifier to construct the menu hierarchical tree.
- The weight of the menu, which is used to identify the position of the menu in the hierarchy.
- The menu block name, which is the group name of the menu. The block name is identical with the content category.

4.3. Categories and Contents Management

The categories and contents management were implemented using the existing class from the ContentExpress module. The categories in the CMS have hierarchical structure, which is also used in grouping the contents.

4.3.1. Category Management

The category management features are covered by a class from the ContentExpress module, which is the `CategoryExpress` class. Following the standard development style of the ContentExpress module, this class contains the functions to create, update, delete, and display the categories. In detail, the functions in this class are:

- The generic functions, which are `api_get`, `api_getall`, and `admin_build_form` functions.

- To create a new category, which is handled by `admin_new`, `admin_create`, and `api_create` functions. The `admin_new` function uses the `admin_build_form` function that constructs the form page where the users fill in the category information, the `admin_create` function handles the data verification and API calling, and the `api_create` is the API function that executes the database insert to the `nuke_ce_categories` table.

- To modify an existing category, which is handled by `admin_edit`, `admin_update`, and `api_update` functions. The `admin_edit` function calls the `api_get` function to retrieve the category record form the database, and calls the `admin_build_form` function to construct the form page where the users can edit the information. The `admin_update` function handles the data verification and API calling. The `api_update` is the API function that processes the database update in the `nuke_ce_categories` table.

- To delete an existing category, which is handled by `admin_delete` and `api_delete` functions. The `admin_delete` function calls the `api_get` function to retrieve the category record form the database, calls the `admin_build_form` function to constructs the form page, provides deletion confirmation page for the users, and also calls the API function. The

`api_delete` is the API function that executes the database delete process in the `nuke_ce_categories` table.

- To display the category hierarchy, which is handled by `admin_view` function. This function calls the `api_getall` function to retrieve all the category records from the database, and constructs the categories based on the hierarchical structure.

The implementation diagram of the `CategoryExpress` class is shown in the Figure 4.5.

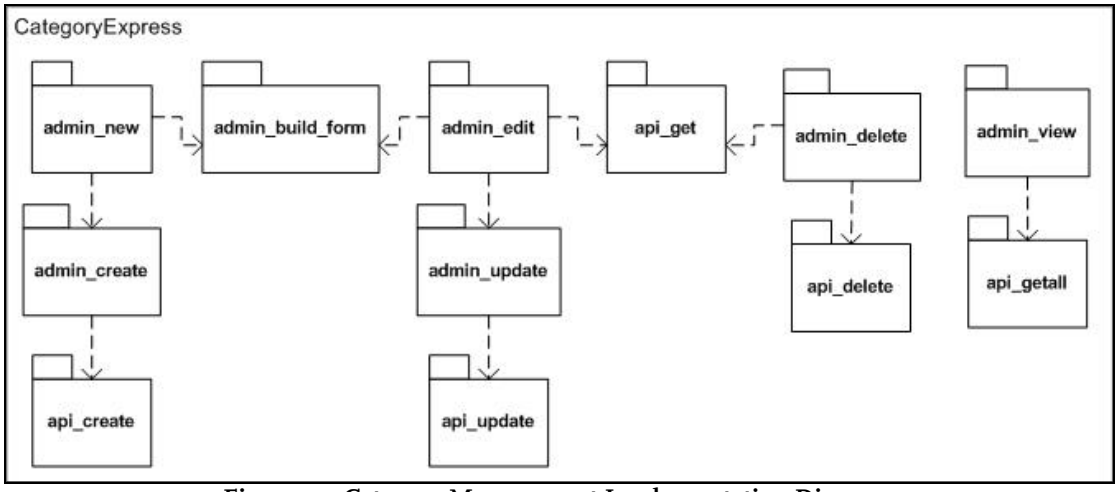


Figure 4.5 Category Management Implementation Diagram

The category has two important variables, which are:

- The category title that is displayed to the users.
- The parent category title, which is used as the identifier to construct the categories hierarchical tree.

4.3.2. Content Management

The content management features are handled by a class from the ContentExpress module, which is the `ContentExpress` class. Following the standard development style of the ContentExpress module, this class contains the functions to create, update, delete, and display the contents.

The default contents listing that provided by the ContentExpress module was a simple flat list. To enhance the usability, the list was changed by grouping the contents based on the category. In the modified CMS, the contents listing are grouped and displayed based on the categories hierarchical tree. This listing style also improves the performance because each database retrieval process only gets a small collection of contents.
Moreover, since the contents are XML files, we developed XSL to display the XML files in standard HTML format.

Another modification regarding the content management feature is the deactivation of content manipulation functions. This is relevant to the project

motivation that we deal with non-proprietary contents, and we will preserve the original content. Thus, although the ContentExpress module provides several content management features, we only use the displaying feature with modification as mentioned before.

The content has several variables, and the most important ones are:

- The content title that is displayed to the users.
- The category title, which is used to group the content by a specific category.
- The content text, which contains the URL to view the original content.
- The content author, which is used to identify the content category during the content import process.
- The content notes, which contains the content unique identifier that used in the import process.

4.4. Metadata Management

The metadata management is the new added feature in the CMS. It is used mostly as backend component to process the metadata information and also to handle features related to the inter-content links. The metadata management functions are placed in one class, which is the `CCMetaExpress` class.

The functions in the `CCMetaExpress` class are:

- `admin_cc_enterbuildlink`, `admin_cc_buildlink`, and `api_cc_buildlink`

  These functions construct the pages related to the inter-content links generation. The `admin_cc_enterbuildlink` function constructs the page where the users define the location path of the file containing the links information that generated by the link recognizer component.
  The `admin_cc_buildlink` function handles the data verification and API calling.
  The `api_cc_buildlink` is the API function that reads the source file and stores the inter-content links information as metadata records in the `nuke_cc_metaitems` table.

- `admin_cc_viewmetadata`

  This function constructs the page displaying the list of inter-content links for a specific content. It is called from the content listing page and uses the content id and the metadata element name to retrieve the metadata records from the `nuke_cc_metaitems` table. The database retrieval is done by the `api_cc_getmetabyceid` function.

- `admin_cc_delmetadata` and `api_cc_delete`

  The `admin_cc_delmetadata` function constructs the confirmation page to delete a metadata record that displayed in the list constructed by the `admin_cc_viewmetadata` function, and calls the API function. The

`api_cc_delete` function is the API function that performs the database delete process in the `nuke_cc_metaitems` table.

In the Figure 4.6 we can see the implementation diagram of the `CCMetaExpress` class.

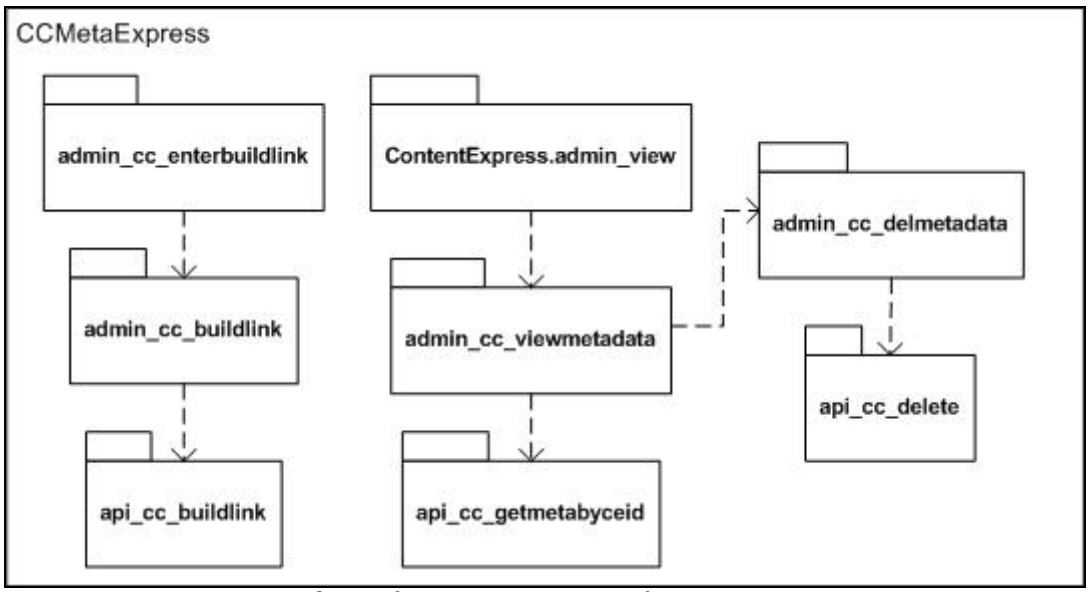


Figure 4.6 Metadata Management Implementation Diagram

4.5. Export Module

The export module was implemented as two sub modules, the content export and the TOC export modules.
The content export module prepares the XML files in the CMS file structure to be indexed. In this preparation, the module merges the metadata as new XML elements in the XML file.
The TOC export module generates the XML TOC file that also used during the indexing process. The TOC is constructed based on the menu hierarchy that stored in the `nuke_me_menuitems` table and the category hierarchy that stored in the `nuke_ce_categories` table.

4.5.1. Content Export

The content export module is defined in the `CCDocExport` class. The main functions in this class are:

- `admin_cc_enterdocexport` and `admin_cc_createdocexport`

  The `admin_cc_enterdocexport` function constructs the page where the users initiate the content export process. In this page the users define the target directory for the export process. The submitted data is processed by the `admin_cc_createdocexport` function, which includes the data validation and the API function calling.

- `api_cc_createdocexport`

  This is the main API function in the content export module. The parameter of this function is the export target directory. The detail execution flow is as follows.

  1. Verify if the target directory parameter is set.
  2. Get the content records from the database. This is filtered by the cc_exported_flag value in the nuke_ce_contentitems table that equal to 0, and also filtered by the content category.
  3. Create the target directory if it does not exist.
  4. Get the CMS storage directory from the nuke_module_vars table.
  5. Loop for each content record retrieved in step 2:
     6. Create sub directories based on the content id or based on the category structure from the nuke_ce_categories table.
     7. Get the file name based on the content records.
     8. Get the content of the file.
     9. Construct additional XML elements based on the metadata retrieved from the nuke_cc_metaitems table.
     10. Update the original file content retrieved in step 8 with the additional elements from step 9.
     11. Write the XML file in the target directory.
     12. Update the cc_exported_flag column in the nuke_ce_contentitems table to 1.
  13. End loop for each content record.
  14. End of the function.

  Most of the steps in the API function, such as the directory creation and the metadata retrieval to the database, are constructed as separate functions in the `CCDocExport` class.

The visual representation of the `CCDocExport` class is as shown in the Figure 4.7. In the Figure 4.7 we can see the functions that are called by the `api_cc_createdocexport` function.

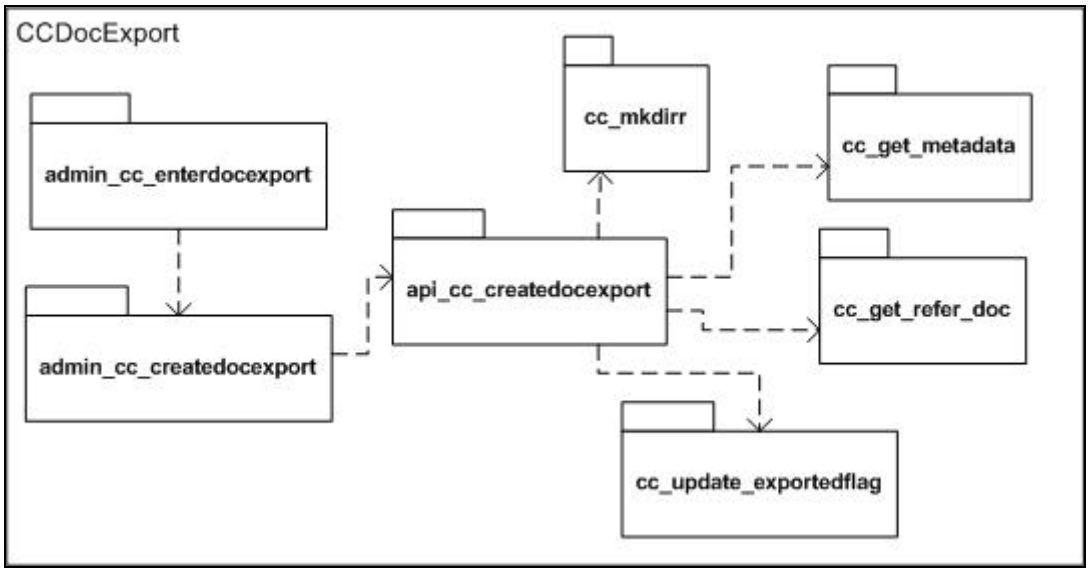


Figure 4.7 Content Export Implementation Diagram

4.5.2. TOC Export

The TOC export module is defined in the `CCManualTOC` class. The main functions in this class are:

- `admin_cc_enterxmltoc` and `admin_cc_createxmltoc`

  The `admin_cc_enterxmltoc` function constructs the page where the users initiate the TOC export process. The parameters for this process are:
  - Target TOC file name.
  - Target directory to store the file.
  - Type of the TOC. The users can choose whether the TOC contains no inter-contents link or it contains the inter-contents links.

  The submitted data is processed by the `admin_cc_createxmltoc` function, which includes the data validation and the API function calling.

- `api_cc_createxmltoc` and `api_cc_createxmltocbycategory`

  This is the main API function in the TOC export module. It retrieves the menu blocks from the `nuke_blocks` table and constructs the TOC sets based on the blocks.

  For each block, it retrieves the menu from the `nuke_me_menuitems` table, hierarchically from top to bottom. For each menu the function checks whether the menu is a menu label or a menu with link to content. For the second case the function constructs the `xlink` attribute to open the content. The function also constructs the sub nodes based on the headers (i.e. chapters, sections, and articles) in the content.
  If the users chose to create the TOC with inter-contents links, the function also retrieves the links information from the metadata (`nuke_cc_metaitems`) table and constructs the sub nodes based on this information.

  Besides the construction based on the menus, the `api_cc_createxmltoc` function calls the `api_cc_createxmltocbycategory` function to construct the TOC sets based on the category hierarchy from the `nuke_ce_categories` table. In general it employs the similar flow as the construction based on the menus, but in this case the function does not need to construct the content headers and the inter-content links sub nodes.

The Figure 4.8 shows the diagram of the `CCManualTOC` class. In the Figure 4.8 we can see the functions that are called by the API functions.

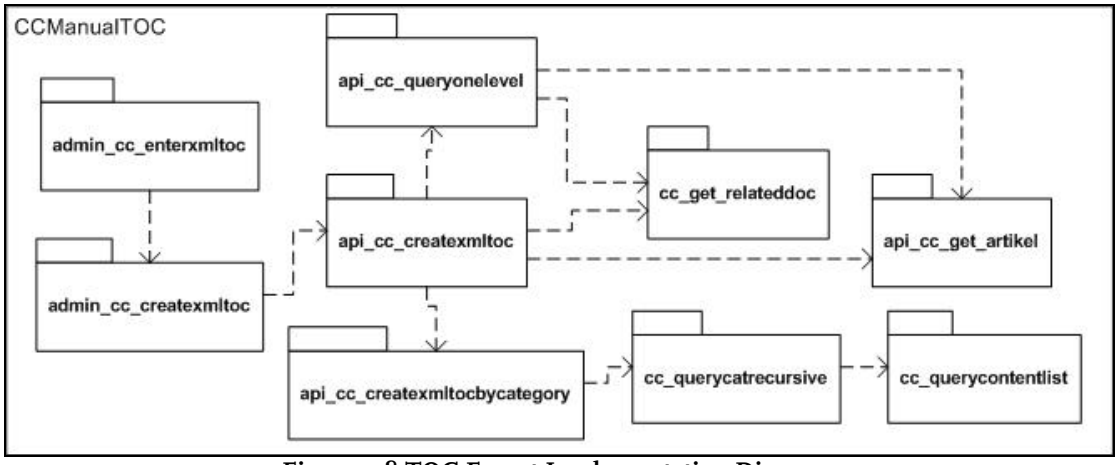


Figure 4.8 TOC Export Implementation Diagram

4.6. Data Population

The important phase during the implementation was the data population. It was the phase where the complete content flow was executed and all the components in the Portal Content Provider system performed their tasks as one integration process. Moreover, the iterative improvement process of the information portal development was based on this populated data.

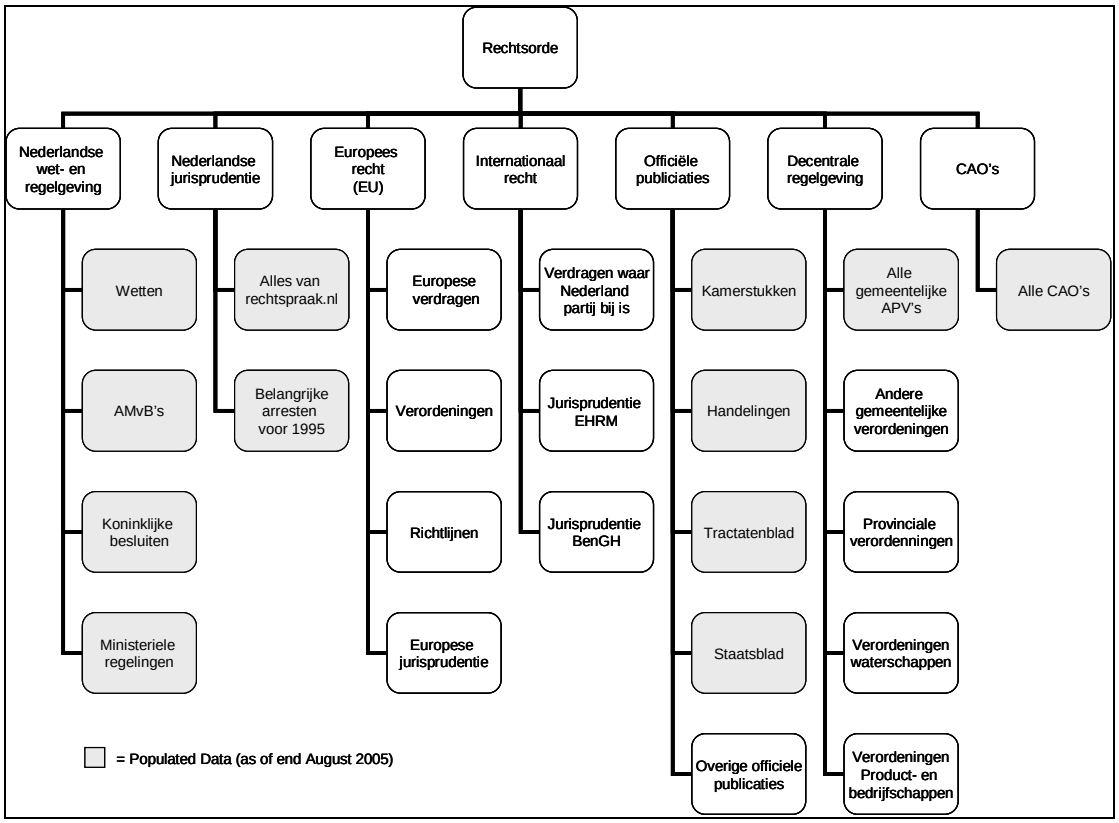


Figure 4.9 Content Categorization

To illustrate the amount of the data we can look at the Figure 4.9. The Figure 4.9 covers the complete set of the contents that will be presented in the portal, grouped into several jurisdiction categories.

The data population was done separately for each content category, which is indicated with grey boxes in the Figure 4.9. As part of the complete content flow, the link recognizer component also generated the inter-contents links between the case laws (in Dutch is the 'jurisprudentie') and the laws (the 'Wetten'). These links were recorded as metadata information of the case laws and presented as two ways referential links between the laws and the case laws in the information portal.

The complete list and amount of the data as of end August 2005 can be found in the following table.

Content Type	**Amount**	**Size**	**TOC Nodes**	**Category Nodes**	**Metadata**
Wetten	1,903	483 MB	2,029	-	-
Jurisprudentie	63,505	1.62 GB	-	4,166	153,449
APV	452	205 MB	480	-	
CAO	475	56.1 MB	772	-	
Officiële Publicaties	171,664	8.04 GB	-	11,627	
AMvB's	448	60.6 MB			
Ministeriële Regelingen	6,089	422 MB			
Total	**244,536**	**10.89 GB**	**3,281**	**15,793**	**153,449**

5. Conclusions and Suggestions for Future Work

This chapter contains a summary of the report and reflects back the result of the project to the research questions defined in Section 1.2. Furthermore it also discusses possible future work related to the CMS and the whole system in general.

5.1. Summary

In this project we have built the complete development life cycle of the CMS that manages the data for the information portal. The project has been initiated by specifying the requirements of the CMS in the scope of the information portal application, as explained in Section 2.1. And as explained in Section 2.2, we have done the analysis phase where we have assessed several open source CMS packages and have selected the PostNuke CMS as the basis implementation framework.

In the architecture and design phase, we have defined the architectures of the Portal Content Provider system and the CMS. In more detail, we have described the database and the site navigation designs of the CMS. All these architectures and designs have been discussed in Chapter 3. In this phase we have identified that we could use the TOC, categories, and contents management functionalities provided by the PostNuke CMS, and we needed to develop the CMS import and export components and also the metadata management.

The development of the new objects and the integration to the PostNuke CMS framework have been done during the implementation phase. In Chapter 4 we have explained the developed PHP classes and objects in the CMS as the realization of the design. During the implementation phase we also populated the content of the CMS. The contents categorization and statistics have been described in Section 4.6.

5.2. Answers for the Research Questions

In the Section 1.2 we formulated the research questions as the basis of this work. Here we discuss the answers for those questions.

We start with the two sub questions and then answer the main question.

Answers for the Sub Questions

The first sub question is:

- How can we use combination of XML and RDBMS technologies to help the users enriching the contents, in this case adding metadata information and constructing inter-contents referential links?

In Section 3.1 we have described the Format Converter component that converts the contents to XML files. The same component also identified the important metadata from the contents and set them as XML elements in the header part of the XML files.
During the CMS import process, the metadata in the XML elements are stored in the metadata table in the RDBMS storage of the CMS and the XML files are stored as the directory structure. Using the provided interface from the CMS, the CMS users can edit the metadata. The design has been discussed in Section 3.2 and the implementation has been described in Section 4.1 and 4.4.

Besides the metadata extracted from the contents, the metadata table in the CMS storage also contains the inter-contents links records that generated by the Links Recognizer. In Section 3.2 has been explained that during the CMS export process the metadata (including the inter-contents links) are merged back to the XML files. These files then are stored and ready to be processed by the Indexing Engine.

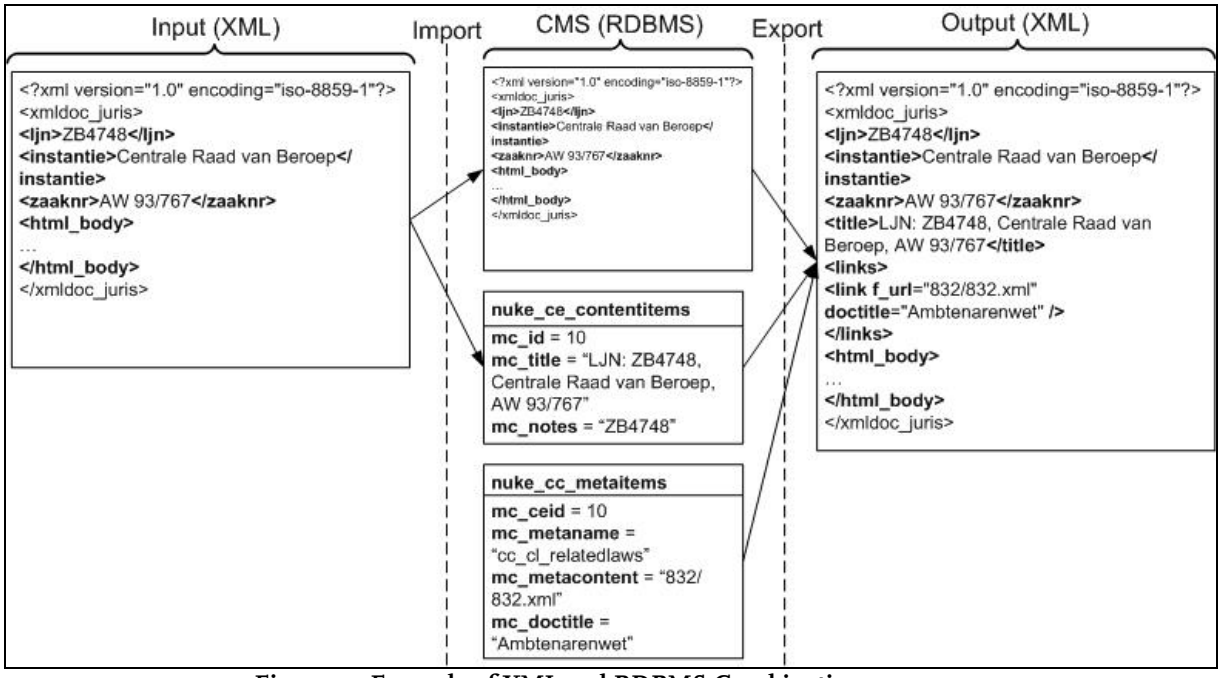


Figure 5.1 Example of XML and RDBMS Combination

As an illustration, we provide an example as shown in the Figure 5.1. The explanations are as the following.

1. We have an XML file as the input of the CMS. The XML file contains `ljn`, `instantie`, and `zaaknr` XML elements that represent metadata of the content.
2. The import module extracts the metadata and stores them in the `nuke_ce_contentitems` table, meanwhile the XML file is stored in the CMS file storage.
3. The Link Recognizer identifies the inter-contents links, and these links are stored in the `nuke_cc_metaitems` table.
4. The export module reads both tables in the CMS database and merges the metadata into the XML file. The output contains the original metadata (i.e.

`ljn`, `instantie`, and `zaaknr`), and also the new metadata (i.e. `title` and `links`).

The second sub question is:

- How can we build the import and export components as the interface between the CMS and other components in the system, which can ensure the consistency and integrity of the content flow?

In terms of consistency, in Section 4.1 we have discussed the implementation of the import module of the CMS. There are two important points there:

- It has been described that during the import process there are two unique variables being defined, the 'author' and 'notes'. The combination of these two variables are used to check if the imported content is new or already exist in the CMS database.
- Moreover, the import module compares the file timestamp with the timestamp of the previous import execution to check if the file is new or already exist at the previous execution.

These methods ensure the consistency of the content in the storage of the Format Converter and the one in the CMS database.

In the same algorithm also has been mentioned that there is a flag column indicating if the content is new or recently updated. This flag is used by the export module, as have been explained in Section 4.5.1, to identify which contents that should be exported. In the end of the export process, those flags are updated to indicate that the contents have been exported successfully.

About the integrity, in Section 4.1 has been mentioned that during the import process the content is linked to the specific node in the TOC based on the title or the unique identifier, or the content is grouped by the content categorization. In this way we integrate the contents from multiple sources into a structure based on the TOC or the category.
In Section 4.5 we have discussed the export process, where these important aspects of the contents are integrated as one collection:

- The inter-content links.
- The contents metadata.
- The TOC.

The inter-content links and the metadata are constructed as XML elements of the XML file representation of the content. The TOC, which contains hierarchical structure of the contents and links to open the XML files, also converted as a XML file. These XML files are stored in one directory and ready to be processed by the Indexing Engine.

Answer for the Main Question

The main research question is:
How can we use open source CMS to maintain non-proprietary contents for an information portal?

To maintain non-proprietary contents for an information portal, we can use open source CMS such as the PostNuke CMS by using it as the development framework and use the basic functionalities (with some modifications) provided by the CMS, such as the menu management to construct the TOC, the metadata management, and the category management.
Additionally, we can develop new components in the CMS, such as the import and export components, to enhance the functionality of the CMS as well as the consistency and integrity of the complete system. The development of these new components employs the development framework and the database design of the PostNuke CMS.

5.3. Possible Future Works

This project can be considered as an early initiative of implementing CMS to maintain non-proprietary contents. There are several aspects related to the CMS and the other components in the system that can be used as main topic for future development or research works.

Intelligence Content Categorization

The categorization in the CMS currently is set by the CMS users, based on the possible law and legislation categorization as explained in Section 4.6. There is a possibility to improve the categorization by identifying the harvested contents and define the category as metadata. This identification can be based on the content source or specific property of the content.
The metadata can be used as a parameter during the CMS import process to create the category structure and group the contents.

Generic Implementation

Currently the project is dedicated for information portal related to the law and legislation and there are several settings that being defined specific for this content type. It can be a further goal to develop a generic CMS that can handle any kind of content type. Using the same architecture, we can develop an application with more user-parameterized variables so it is independent from the content type.

Automated Integrated Background Processing

As we have seen in the complete architectural view in Section 3.1, there are several related components in the system. Currently those components execute their tasks independently and each execution flow depends on the user action. It is a possible improvement to create a scheduler component that can automate the execution as a background process and trigger a notification to the user if there is a necessity to have human involvement in the workflow, or if there is unexpected result during the execution.

Bibliography

[ATK, 2003] L. Atkinson and Z. Suraski, 2003, **Core PHP Programming**, 3rd Edition, Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, USA.

[CER, 2002] S. Ceri, P. Fraternali, and M. Matera, 2002, **Conceptual Modeling of Data-Intensive Web Applications**, in *IEEE Internet Computing 6, No. 4 (July-August 2002)*, pp. 20-30.

[EDE, 2003] A. H. Eden and R. Kazman, 2003, **Architecture, Design, Implementation**, in *Proceedings of the 25th International Conference on Software Engineering (ICSE 25)*, Portland, OR, USA, 3-10 May, pp. 149-159.

[KAN, 1999] K. C. Kang, S, G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, 1990, **Feature-Oriented Domain Analysis (FODA) Feasibility Study**, *Technical Report CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[MAC, 2001] R. Mack, Y. Ravin, and R. J. Byrd, 2001, **Knowledge Portals and the Emerging Digital Knowledge Workplace**, in *IBM Systems Journal Vol. 40 No. 4*, pp. 925-955.

[NAK, 2002] R. Nakano, 2002, **Web Content Management: A Collaborative Approach**, Addison-Wesley, Boston, MA, USA.

[PRI, 2003] T. Priebe and G. Pernul, 2003, **Towards Integrative Enterprise Knowledge Portals**, in *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM 2003)*, New Orleans, LA, USA, 3-8 November, pp. 216-223.

[ROB, 2003] J. Robertson, 2003, **So, What is a Content Management System?**, Step Two Design Pty. Ltd..

[VID, 2001] R. Vidgen, S. Goodwin, and S. Barnes, 2001, **Web Content Management**, in *Proceedings of the 14th International Electronic Commerce Conference*, Bled, Slovenia, 25-26 June, pp. 465-480.

[WIE, 2003] Karl E. Wiegers, 2003, **Software Requirements**, 2nd Edition, Microsoft Press, Redmond, WA, USA.

Appendix A - Technical Documentation

List of classes in the PostNuke CMS, specifically in the ContentExpress module, that relevant with the works in the project:

- `BaseExpress`
- `CategoryExpress`
- `CCDocExport`
- `CCImportXML`
- `CCManualTOC`
- `CCMetaExpress`
- `ContentExpress`
- `InitialExpress`
- `MenuExpress`

BaseExpress

File name: `/pnclass/BaseExpress.php`

Description: The base class of the ContentExpress module. This class is an inheritance of the `InitialExpress` class. This class contains several functions for debugging purpose.
Note: This class is not directly used in the development.

CategoryExpress

File name: `/pnclass/CategoryExpress.php`

Description: The class related to the category management. This class inherits the `BaseExpress` class.

Methods:

- `admin_build_form`
  Function to build content form.
- `admin_create`
  Function to process a new category creation.
- `admin_delete`
  Function to delete an existing category.
- `admin_edit`
  Function to build form to modify an existing category.
- `admin_new`
  Function to build form to create a new category.
- `admin_qupdate`
  Function to process quick update on existing category.
- `admin_update`
  Function to process an existing category modification.
- `admin_view`
  Function to build hierarchical views of the existing categories.
- `api_create`

API function to insert new row in category table while creating a new category.

- `api_delete`
  API function to delete a row in category table while deleting an existing category.
- `api_get`
  API function to retrieve a row from category table.
- `api_getall`
  API function to retrieve all rows from category table.
- `api_getsubcategories`
  API function to retrieve all rows from category table based on `parent_id` parameter.
- `api_qupdate`
  API function to update `parent_id` column in the category table while processing the quick update.
- `api_update`
  API function to update columns in the category table while processing category modification.

CCDocExport

File name: `/cc_export_xml/CCDocExport.php`

Description: The class contains functions related to export module. This class inherits the `BaseExpress` class.

Methods:

- `admin_cc_createdocexport`
  The content export process handler function. It is called by `admin_cc_enterdocexport` function and it passes the request to the `api_cc_createdocexport` API function.
- `admin_cc_enterdocexport`
  This is the main window of content export process. The users enter the target directory name and the request will be processed by `admin_cc_createdocexport`.
- `api_cc_createdocexport`
  The main API function to export the contents from the CMS. It reads the CMS directory and creates XML file in the target directory. This function will call itself recursively to export contents based on category.
- `cc_copy_dir`
  Function to recursively copy a directory and its contents. It is called by the API function.
- `cc_copy_meta_files`
  The function to copy metadata files during content export process. It is called by the API function.
- `cc_copy_related_files`
  The function to copy related files during content export process. Currently it is used to copy PDF and image files (for APV and CAO). It is called by the API function.

- `cc_get_category_data`
  The function to get the category data for specific official publication. It is called by the API function.
- `cc_get_menu_titles`
  The function to get the menu titles for specific law. It is called by the API function.
- `cc_get_metadata`
  The function to get metadata information related to the exported content. It is called by the API function.
- `cc_get_parent_title`
  The function to get the province name for specific APV, or the CAO name for specific AVV. It is called by the API function.
- `cc_get_refer_doc`
  The function to construct the referring content in laws (including in article levels). It is called by the API function.
- `cc_mkdirr`
  Function to create recursive directories structure. It is called by the API function.
- `cc_update_exportedflag`
  The function to update the export flag in the `nuke_ce_contentitems` table. It is called by the API function.

CCImportXML

File name: `/cc_import_xml/CCImportXML.php`

Description: The class contains functions related to import module. This class inherits the `BaseExpress` class.

Methods:
- `admin_cc_enterrunimport`
  The second screen of the import process. This function is called by `admin_cc_enterrunimporttype` function. This function calls `admin_cc_generaterunimport` function.
- `admin_cc_enterrunimporttype`
  The first screen of the import process. The users select the content type to be imported and the process will be transferred to `admin_cc_enterrunimport`.
- `admin_cc_generaterunimport`
  This function is called by the import screen (`admin_cc_enterrunimport`). This function performs variables validation and calls the import API function.
- `admin_cc_matchtoctitle`
  The function to handle the linking process between contents and TOC. It is called by `admin_cc_matchtoctitletype` and it calls the API function.
- `admin_cc_matchtoctitletype`
  The main window of the linking process between contents and TOC. The users select the content category that will be linked, and the process will be transferred to `admin_cc_matchtoctitle`.

- `admin_cc_modifyconfig`
  The second screen of import configuration modification. This function is called by `admin_cc_modifyconfigtype`. This function calls `admin_cc_updateconfig`.
- `admin_cc_modifyconfigtype`
  The first modify import configuration screen. The users select the content type and the process will be transferred to `admin_cc_modifyconfig`.
- `admin_cc_updateconfig`
  This function is called by the import configuration modification screen (`admin_cc_modifyconfig`). This function validates and updates the import parameters.
- `api_cc_importcontent`
  The main API function to import the contents to the CMS. It reads the source directory and copies the recent files to the CMS directory. Then it stores the information in the `nuke_ce_contentitems` table.
- `api_cc_matchtoctitle`
  The API function to link the contents with TOC. The linking process is based on the reference columns in the database.
- `cc_copy_dir`
  Function to recursively copy a directory and its contents. It is called by the API function.
- `cc_create_category`
  Function to create category structure based on source directory structure.
- `cc_get_category_id`
  Function to get category ID with category title as input parameter.
- `cc_get_category_title`
  Function to get category title with category ID as input parameter.
- `cc_get_dirname`
  Function to get recursively the directory names under specific root directory.
- `cc_get_dirtitle`
  Function to get title based on a reference file.
- `cc_get_filename`
  Function to get file names under a directory.
- `cc_get_metadata`
  Function to read metadata elements of a specific content.
- `cc_get_top_categories`
  Function to get top level categories, used to construct the categories selection list.
- `cc_modify_menuitem`
  Function to modify reference column in the menu table, as the result of the linking between contents and TOC process.

CCManualTOC

File name: `/cc_export_xml/CCManualTOC.php`

Description: The class contains functions related to TOC export process. This class inherits the `BaseExpress` class.

Methods:

- `admin_cc_createxmltoc`
  The manual TOC generation process handler function. It is called by `admin_cc_enterxmltoc` function and it passes the request to the API function `api_cc_createxmltoc`.
- `admin_cc_enterxmltoc`
  The main window of manual TOC generation process. The users enter the target directory name and target file name and the request will be processed by `admin_cc_createxmltoc`.
- `api_cc_createxmltoc`
  The main API function to generate the manual TOC. It uses the menu structure to construct manual TOC as XML file. It calls `api_cc_queryonelevel` to construct nested structure. The file result will be stored in the target directory.
- `api_cc_createxmltocbycategory`
  Function to construct manual TOC based on category structure. It reads the category structure in the CMS and creates the TOC nodes. This function is called by `api_cc_createxmltoc`. It calls the `cc_querycatrecursive` function to construct nested TOC nodes.
- `api_cc_get_artikel`
  Function to construct document headers as TOC nodes. It reads the documents and extracts the header based on html header tags.
- `api_cc_queryonelevel`
  The function to construct nested TOC elements. It reads the menu structure to construct TOC nodes, and run in self-loop until reaches the lowest menu level. This function is initially called by `api_cc_createxmltoc`.
- `cc_construct_one_h2`
  The function to construct TOC nodes for document header level. It used for the situation if exist one h2 header and article header. This function is internally used by `api_cc_get_artikel`.
- `cc_construct_one_h2_in_boek`
  The function to construct TOC nodes for document header level. It used for the situation if exist one h2 header and article header in a boek header. This function is internally used by `api_cc_get_artikel`.
- `cc_construct_two_h2`
  The function to construct TOC nodes for document header level. It used for the situation if exist two h2 headers and article header. This function is internally used by `api_cc_get_artikel`.
- `cc_construct_two_h2_in_boek`
  The function to construct TOC nodes for document header level. It used for the situation if exist two h2 headers and article header in a boek header. This function is internally used by `api_cc_get_artikel`.
- `cc_construct_two_h2_in_toph2`
  The function to construct TOC nodes for document header level. It used for the situation if exist three h2 headers and article header. This function is internally used by `api_cc_get_artikel`.
- `cc_gen_array_el`

The function to set the document block as array. It reads the document block, extract the title and the reference id, and construct them as array. This function is internally used by `api_cc_get_artikel`.

- `cc_get_anchor_id`
  The function to get document anchor id to construct ref link.
- `cc_get_relateddoc`
  The function to identify referring documents and then construct TOC sub nodes based on the document information. This function is called by several functions that construct nodes in manual TOC.
- `cc_querycatrecursive`
  The function to construct nested TOC elements based on category. It reads the category structure to construct TOC nodes, and run in self-loop until reaches the lowest category level. This function is initially called by `api_cc_createxmltocbycategory`.
- `cc_querycontentlist`
  The function to construct lowest nested TOC elements based on category. It reads the content under specific category level and constructs the TOC sub nodes. This function is called by `cc_querycatrecursive`.
- `cc_trim_ahref`
  This is the function to remove href tag from the header title. This function is internally used by `api_cc_get_artikel`.

CCMetaExpress

File name: `/cc_import_xml/CCMetaExpress.php`

Description: The class contains functions related to metadata management. This class inherits the `BaseExpress` class.

Methods:

- `admin_cc_buildlink`
  The function to handle the inter-content link generation process. It is called by `admin_cc_enterbuildlink` and it calls the API function `api_cc_buildlink`.
- `admin_cc_delmetadata`
  The function to construct delete metadata window. It shows confirmation page to the users, and then calls the API function `api_cc_delete`.
- `admin_cc_enterbuildlink`
  The function to construct the inter-content link generation window. It calls `admin_cc_buildlink` function as the process handler
- `admin_cc_viewmetadata`
  The function to construct the metadata list window.
- `api_cc_buildlink`
  The main API function to generate inter-content links. It reads the link reference file name and insert/update the `nuke_cc_metaitems` table.
- `api_cc_create`
  The main API function to create metadata record in `nuke_cc_metaitems` table
- `api_cc_delete`

The main API function to delete metadata record in `nuke_cc_metaitems` table.

- `api_cc_getmetabyceid`
  The main API function to get metadata record in `nuke_cc_metaitems` table based on the content ID.
- `api_cc_getmetabynameandcontent`
  The main API function to get metadata record in `nuke_cc_metaitems` table based on the metadata name.
- `api_cc_update`
  The main API function to update metadata record in `nuke_cc_metaitems` table.

ContentExpress

File name: `/pnclass/ContentExpress.php`

Description: The class contains functions related to document management. This class inherits the `BaseExpress` class.

Methods:

- `admin_build_form`
  Function to build content form.
- `admin_create`
  Function to process a new content creation.
- `admin_delete`
  Function to delete an existing content.
- `admin_edit`
  Function to build form to modify an existing content.
- `admin_modifyconfig`
  Function to modify configuration variables related to the content.
- `admin_new`
  Function to build form to create a new content.
- `admin_qupdate`
  Function to process quick update on existing content.
- `admin_update`
  Function to process an existing content modification.
- `admin_updateconfig`
  Function to process the content configuration modification.
- `admin_view`
  Function to build hierarchical views of the existing contents.
- `api_buildindex`
  Function to show statistics related to a specific content.
- `api_create`
  API function to insert new row in content table while creating a new content.
- `api_delete`
  API function to delete a row in content table while deleting an existing content.
- `api_get`

API function to retrieve a row from content table.
- `api_getall`
  API function to retrieve all rows from content table.
- `api_getall_buildlist`
  API function to construct content list from all rows in the content table.
- `api_getallchild`
  API function to get children of a specific content.
- `api_getbycatid`
  API function to get contents based on specific category.
- `api_getlayouts`
  API function to get all records from layout table.
- `api_getstatuses`
  API function to get all records from status table.
- `api_qupdate`
  API function to update `parent_id` column in the content table while processing the quick update.
- `api_update`
  API function to update columns in the content table while processing content modification.
- `api_updateread`
  API function to update number of read statistic of a specific content.
- `display_content`
  Function to display a single content.
- `user_cc_viewhtmlfromxml`
  Function to display a XML content using appropriate XSL.
- `user_display`
  Function to display a single content.
- `user_mailfriend`
  Function to process if the users send a specific content via e-mail.
- `user_mailsent`
  Function to construct acknowledgement screen after sending process.
- `user_print`
  Function to print a single content.
- `user_sendfriend`
  Function to construct main screen for the users to send a specific content via e-mail.

InitialExpress

File name: `/pnclass/BaseExpress.php`

Description: The initial class of the ContentExpress module.
Note: This class is not directly used in the development.

MenuExpress

File name: `/pnclass/MenuExpress.php`

Description: The class contains functions related to TOC (menu) management. This class inherits the `BaseExpress` class.

Methods:

- `admin_build_form`
  Function to build content form.
- `admin_create`
  Function to process a new menu creation.
- `admin_delete`
  Function to delete an existing menu.
- `admin_edit`
  Function to build form to modify an existing menu.
- `admin_menublock_select_form`
  Function to build screen containing menu blocks selection list.
- `admin_modifyconfig`
  Function to modify configuration variables related to the menu.
- `admin_new`
  Function to build form to create a new menu.
- `admin_qupdate`
  Function to process quick update on existing menu.
- `admin_update`
  Function to process an existing menu modification.
- `admin_updateconfig`
  Function to process the menu configuration modification.
- `admin_view`
  Function to build hierarchical views of the existing menus.
- `api_create`
  API function to insert new row in menu table while creating a new menu.
- `api_delete`
  API function to delete a row in menu table while deleting an existing menu.
- `api_get`
  API function to retrieve a row from menu table.
- `api_getall`
  API function to retrieve all rows from menu table.
- `api_qupdate`
  API function to update `parent_id` column in the menu table while processing the quick update.
- `api_update`
  API function to update columns in the menu table while processing menu modification.
- `build_cache`
  Function to construct menu structure caching for JavaScript menu display.
- `build_menublock_tree`
  Function to construct menu hierarchy grouped by specific menu blocks.
- `user_menu`
  Function to call specific content from a referring menu.

Appendix B - CMS User Guide Documentation

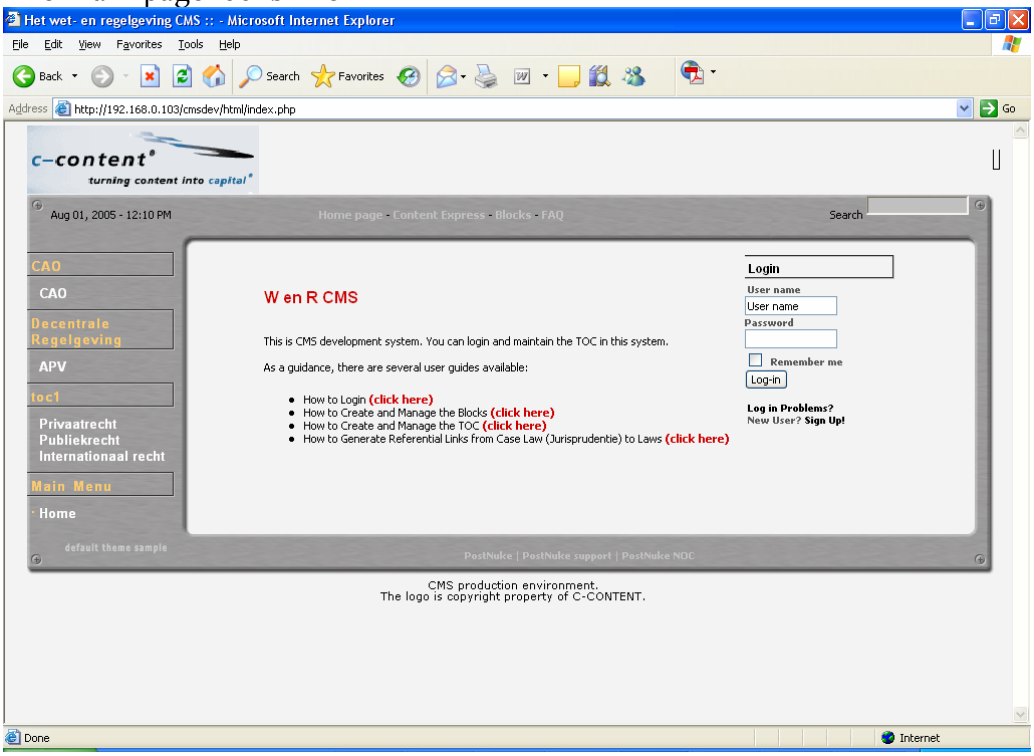
Contents:

1. Login.
2. Create and manage layout blocks.
3. Create and manage the TOC.
4. Create and manage the category.
5. Configure and execute import process.
6. Generate referential links from case laws (jurisprudentie) to appropriate laws.
7. Generate links between contents and the TOC.
8. Execute export process.

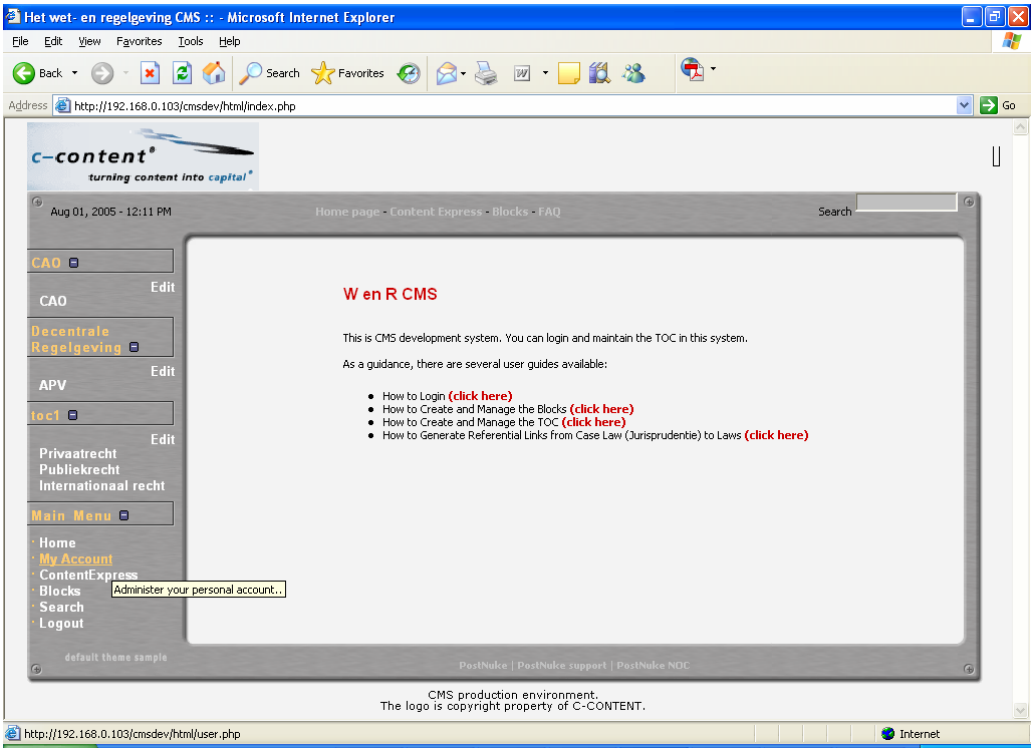
Login

To access the CMS and maintain the TOC, you need to login to the CMS. The steps are:

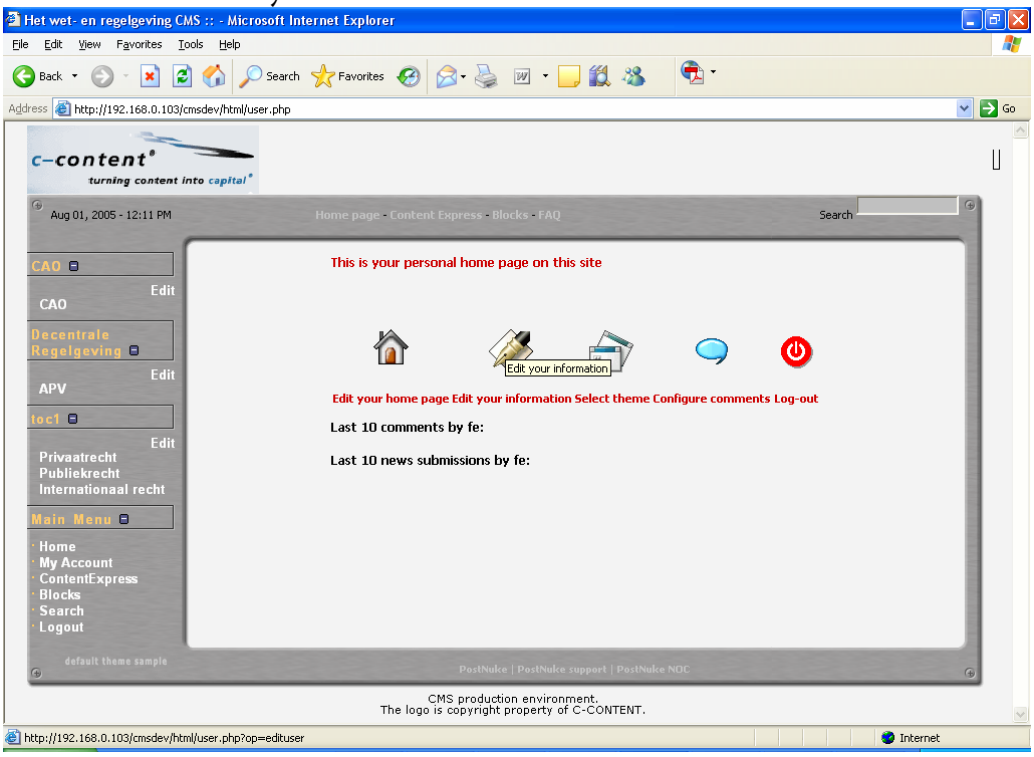
1. The CMS is accessible at http://legal01/cmsprod/html/index.php. For easy access, you can bookmark the link to your web-browser favorite.
2. The main page looks like



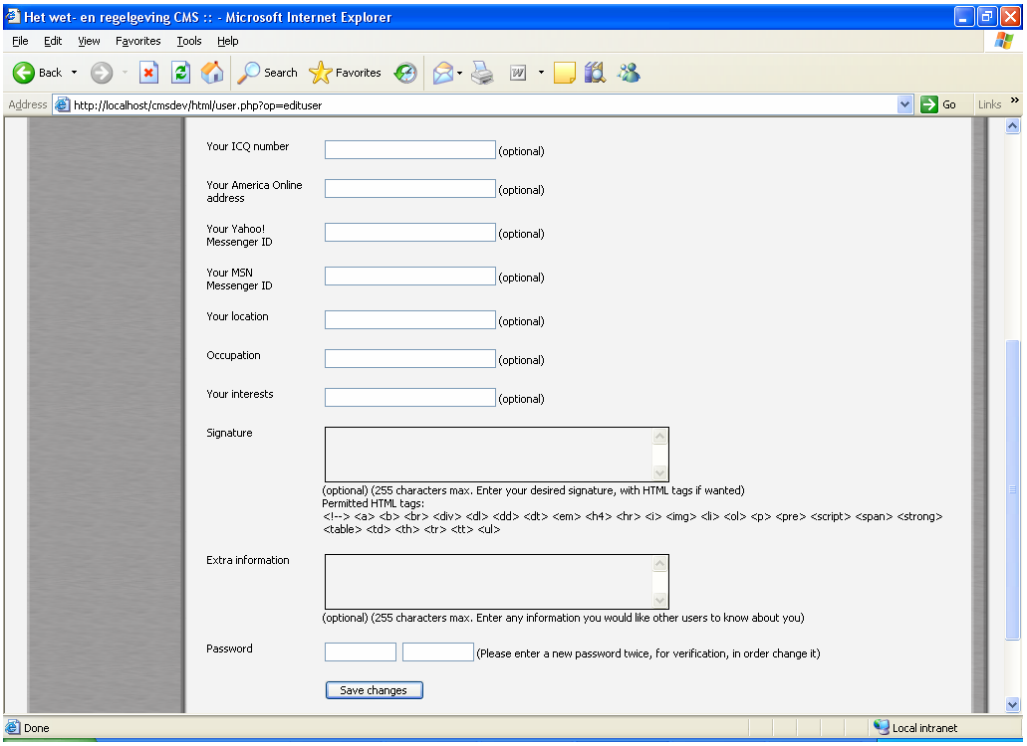
3. You can login using the same user as the network user name. The default password is the same as your user name.
4. You can change your password by navigating to "**My Account**".



5. Then click on "**Edit your information**".



6. The last field is the password field, where you can type in your new password, confirm it, and click Submit button.

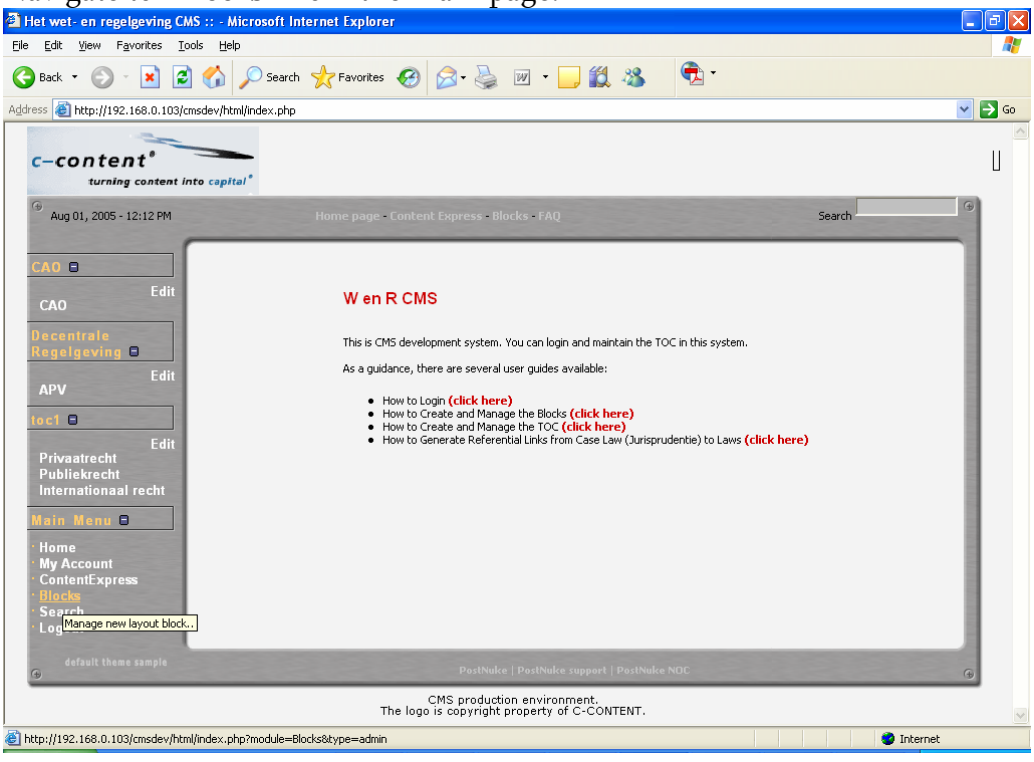


Create and Manage Layout Blocks

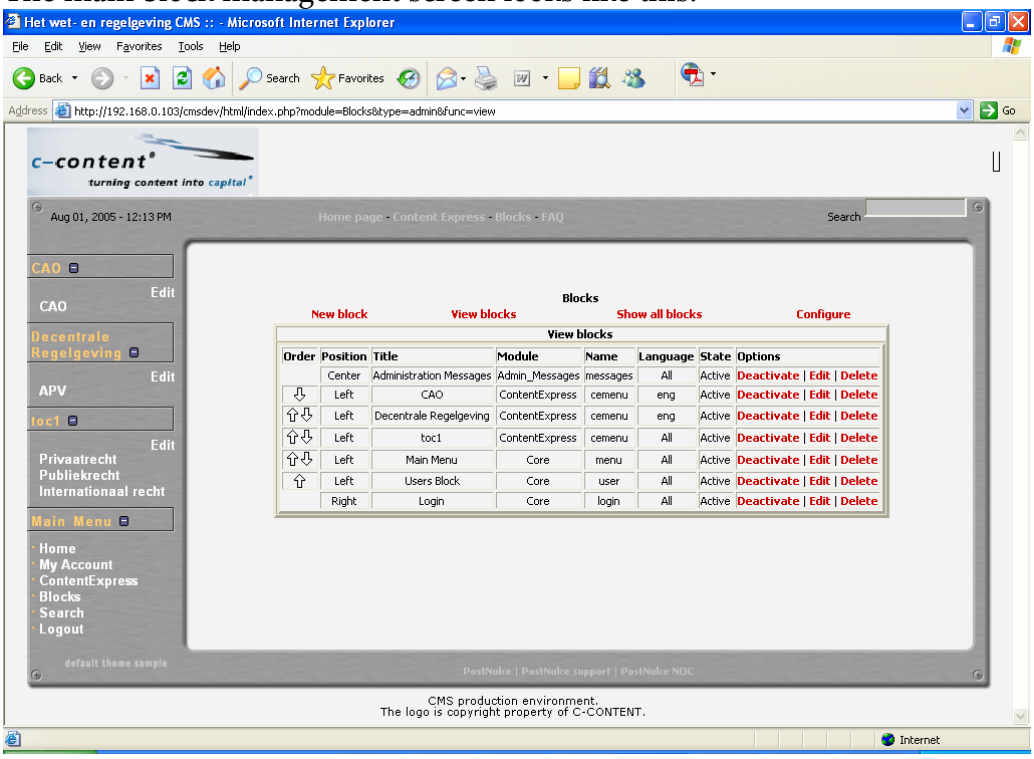
For presentation purpose for the users, we need to set **blocks** for specific user group. For example: a block for lawyers, a block for law student, and a block for public. Each block contains different TOC set, which is appropriate to the user group.

To create a new block, the steps are:

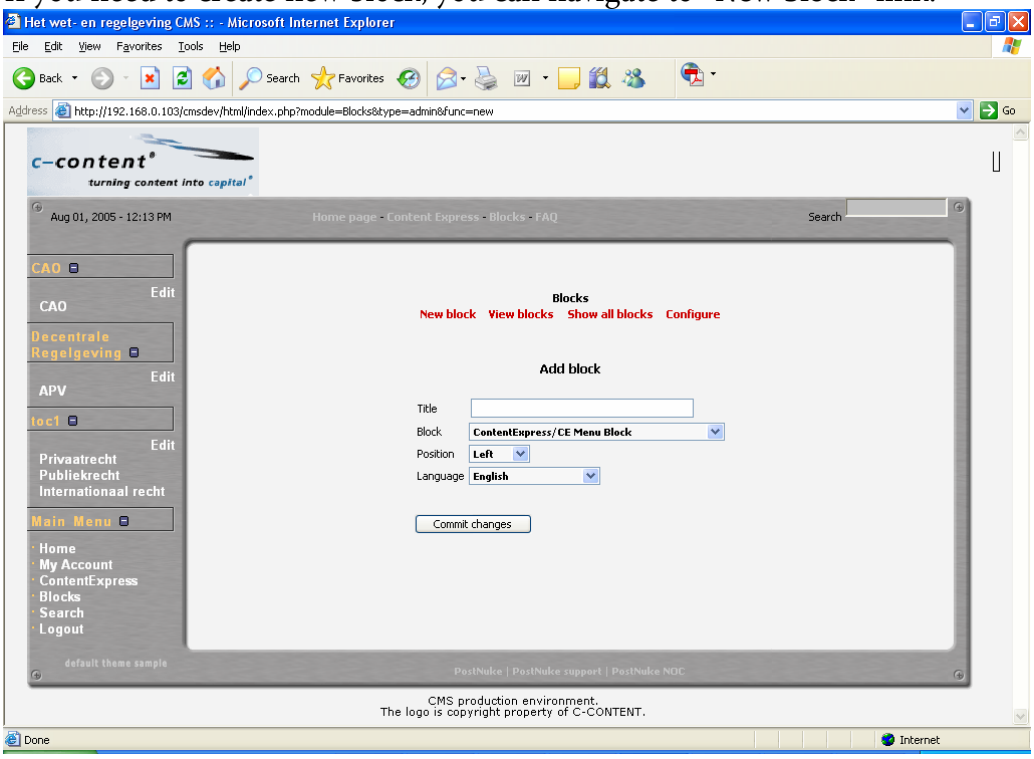
1. Navigate to "**Blocks**" from the main page.



2. The main block management screen looks like this:



3. By default you have block **TOC1** to be used.
4. If you need to create new block, you can navigate to "**New block**" link.

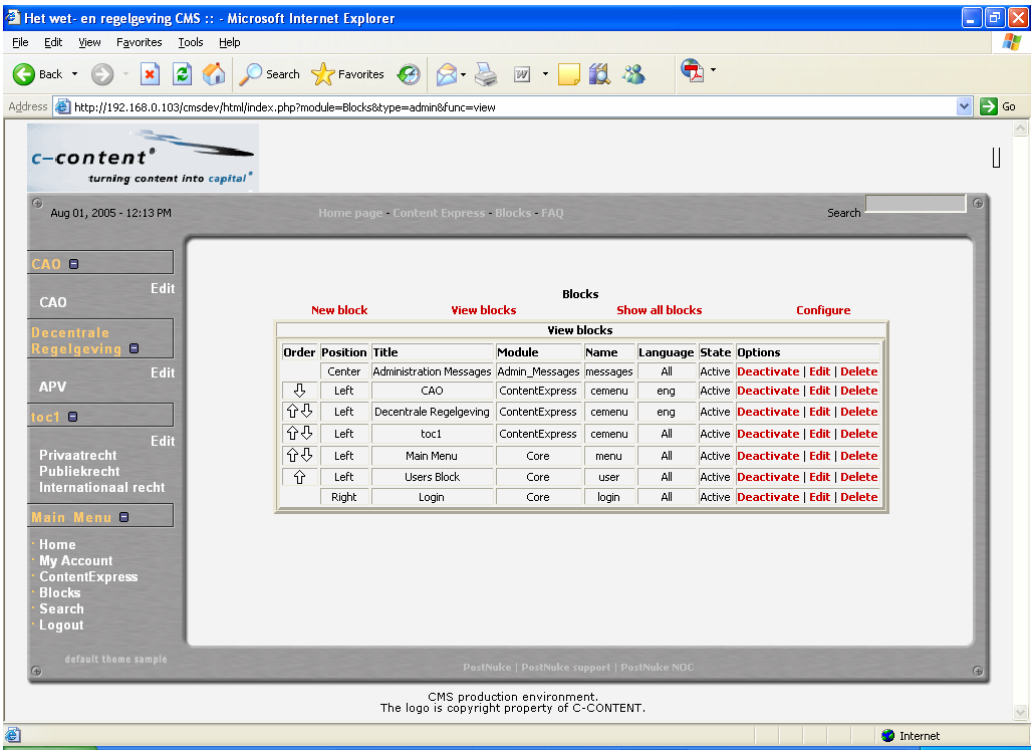


5. You need to fill in/select from list for these fields:
    - **Title**: Enter your block name, for example 'Lawyers', 'Students', 'Public'.

- **Block**: Always select "**ContentExpress/CE Menu Block**".
- **Position**: The viewing position. I suggest putting it in "**Left**".
- **Language**: The default language, no effect for current time. You can use "**English**" as default language.

Then click the "**Commit changes**" button to create the new block.

You can ignore the following screen, and navigate to "View blocks" to view the blocks list.

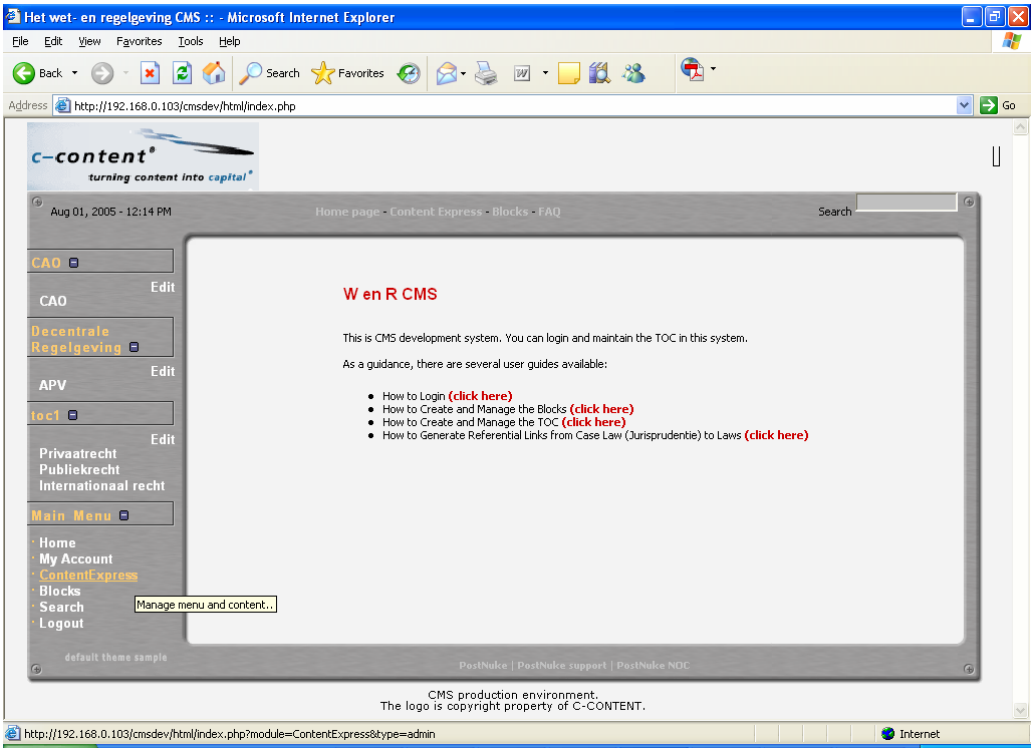


6. You can **Edit** the existing blocks to change the **Position** and **Language**.
7. You can **Deactivate** the blocks that you don't need anymore. Later on you can again **Activate** these blocks.
8. You also can **Delete** the unused blocks. **Note: Don't delete any blocks.** If you need to remove a block, use **Deactivate** instead of **Delete.**

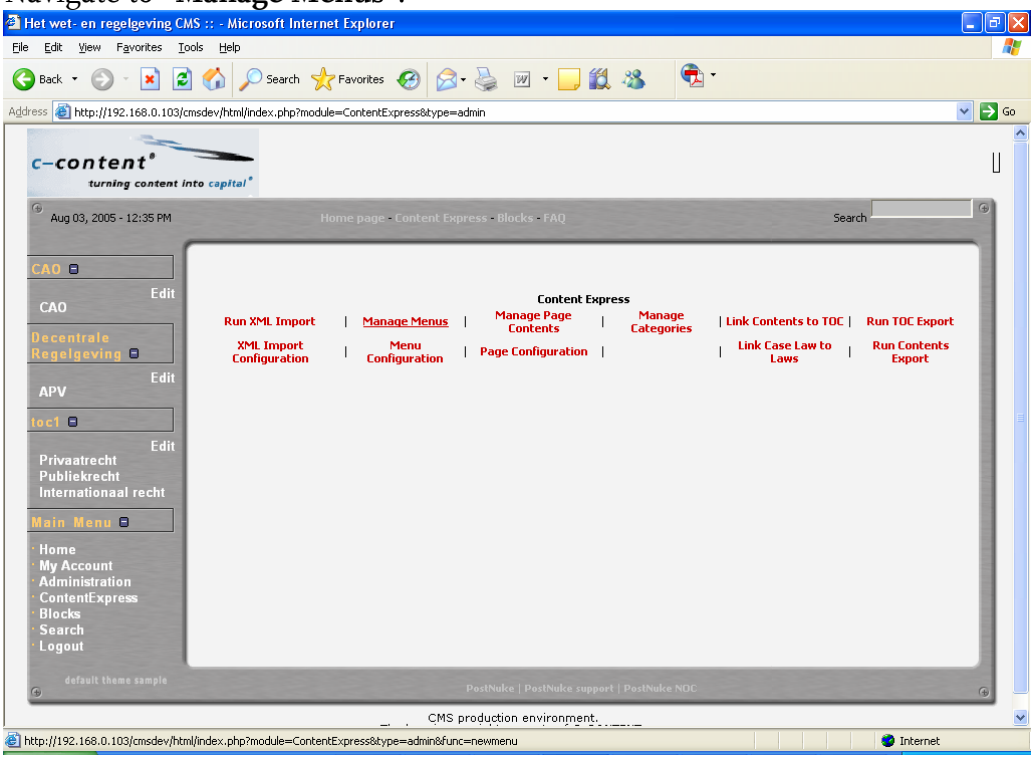
Create and Manage the TOC

After you create the layout blocks, you can start filling in the TOC hierarchy. The steps are:

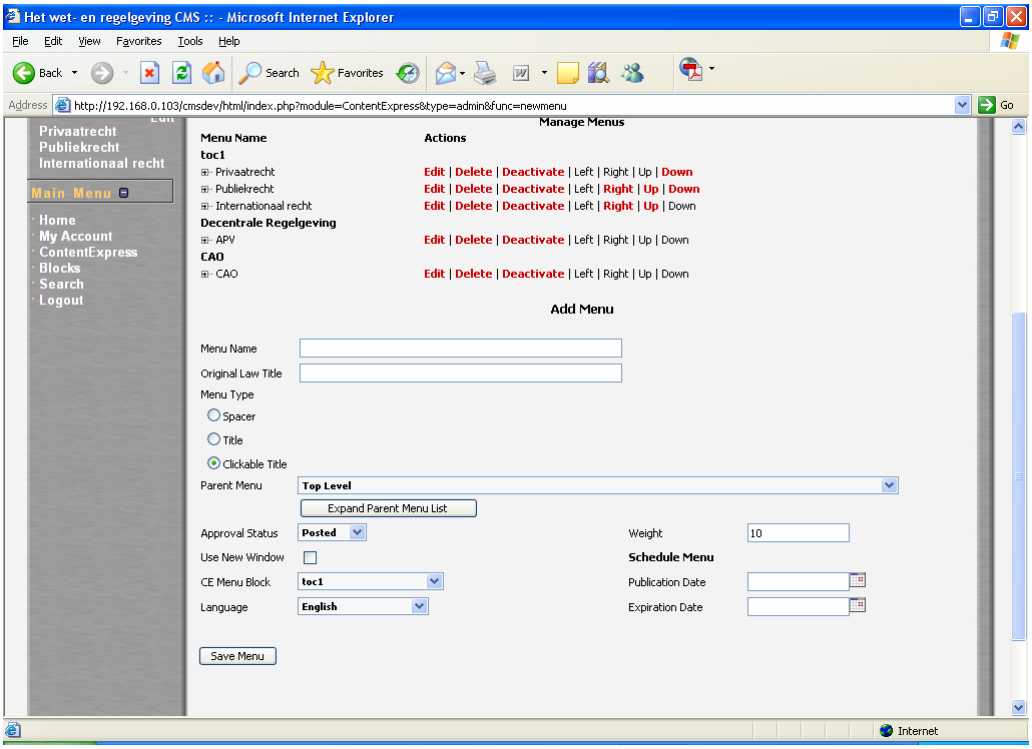
1. Navigate to "**ContentExpress**" from the main menu.



2. Navigate to “**Manage Menus**”.

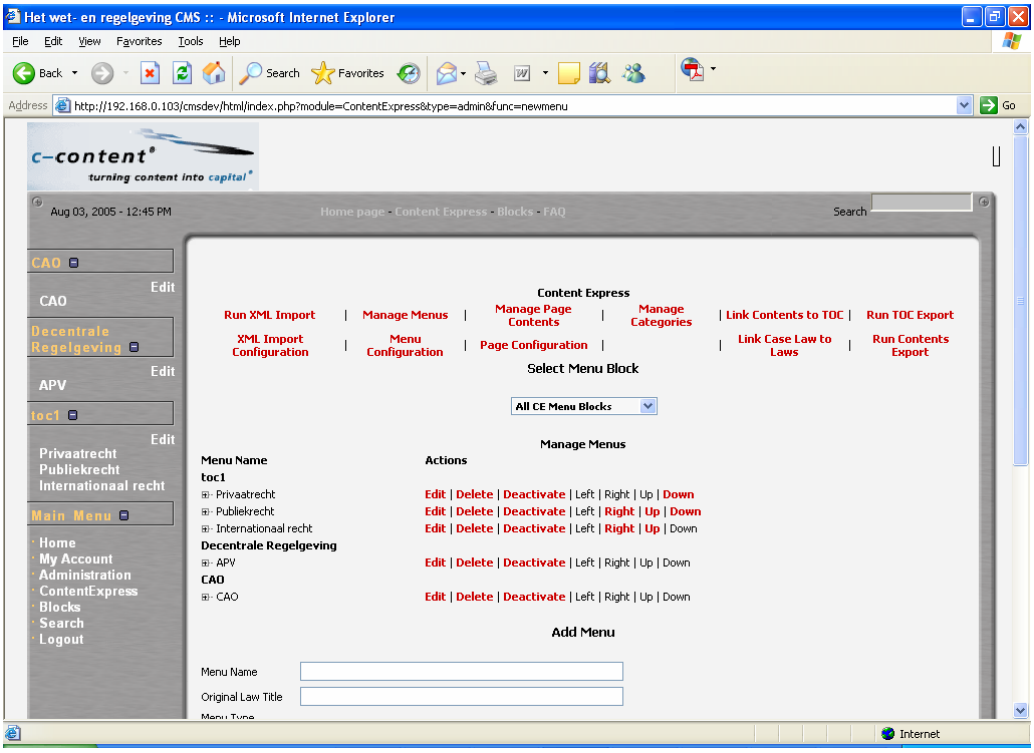


3. The main menu management screen looks like:



4. To create the TOC set, you start with the top level. You need to fill in these fields to create the top level menu:
    - **Menu Name**: The name of the top level menu.
    - **Original Law Title**: The title of the original law document that will be linked to the menu.
    - **Menu Type**: Always choose "**Clickable Title**" (default).
    - **Parent Menu**: Choose "**Top Level**" for top level menus.
    - **Approval Status**: Choose "**Posted**" (default) to directly make the menus viewable to the users. Otherwise choose "**Preview**" (you can only see the result from the ContentExpress management screen).
    - **Use New Window**: Always left blank.
    - **CE Menu Block**: Choose the block name in which this menu belongs to. The blocks are the layout block you defined in previous step.
    - **Language**: Use English as default language.
    - **Weight**: The order of the menu. I suggest you to use 100 as the first order, then 200, 300, 400, and so on. If you later on need to insert a menu in between, you can use a number in between for the new weight (for example 250 between 200 and 300).
    - **Publication Date**: Leave it blank.
    - **Expiration Date**: Leave it blank.

    Click the "**Save Menu**" button to save your new menu.
5. Once you have the top level menus, you can add more menus that belong to specific top level menus. To do this you can follow the step no. 4 above, but choose the appropriate parent menu in the **Parent Menu** selection list.
6. From the main ContentExpress screen you can also edit the existing menus.

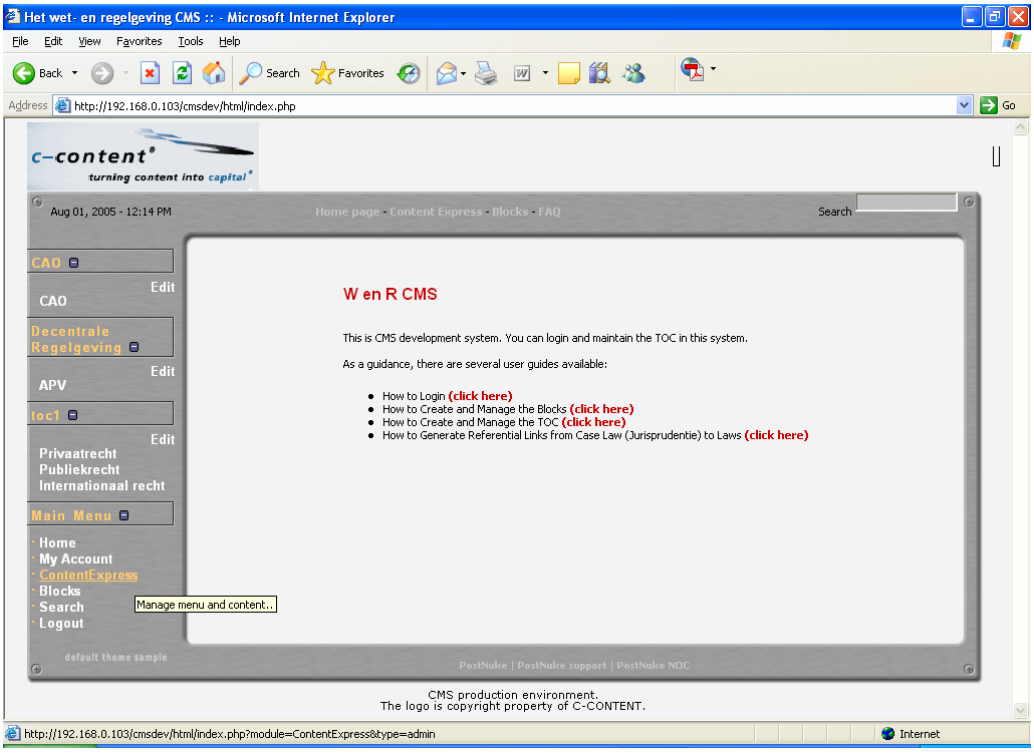


7. You can **Edit** the menu, and change the entries of the menu (the same fields as if you create a new menu).
8. You can **Delete** a menu, if it is unused.
9. You can **Deactivate** a menu so the menu will not be visible to the users. Later on you can **Activate** the menu again.
10. You can move the position of a menu in the hierarchy by moving it:
    - Left: Move a menu to be a parent level (one level above).
    - Right: Move a menu to be a child level (one level below).
    - Up: Move the position of a menu up, but still in the same level.
    - Down: Move the position of a menu down, but still in the same level.

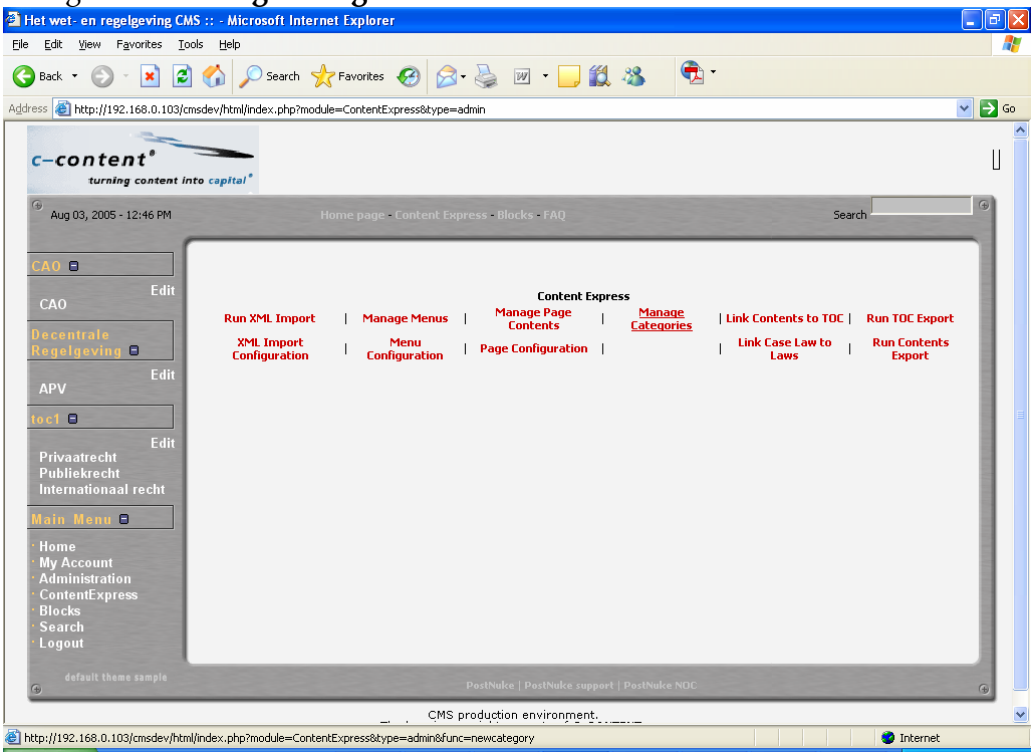
Create and Manage the Category

The contents are grouped by category. To create a category, the steps are:

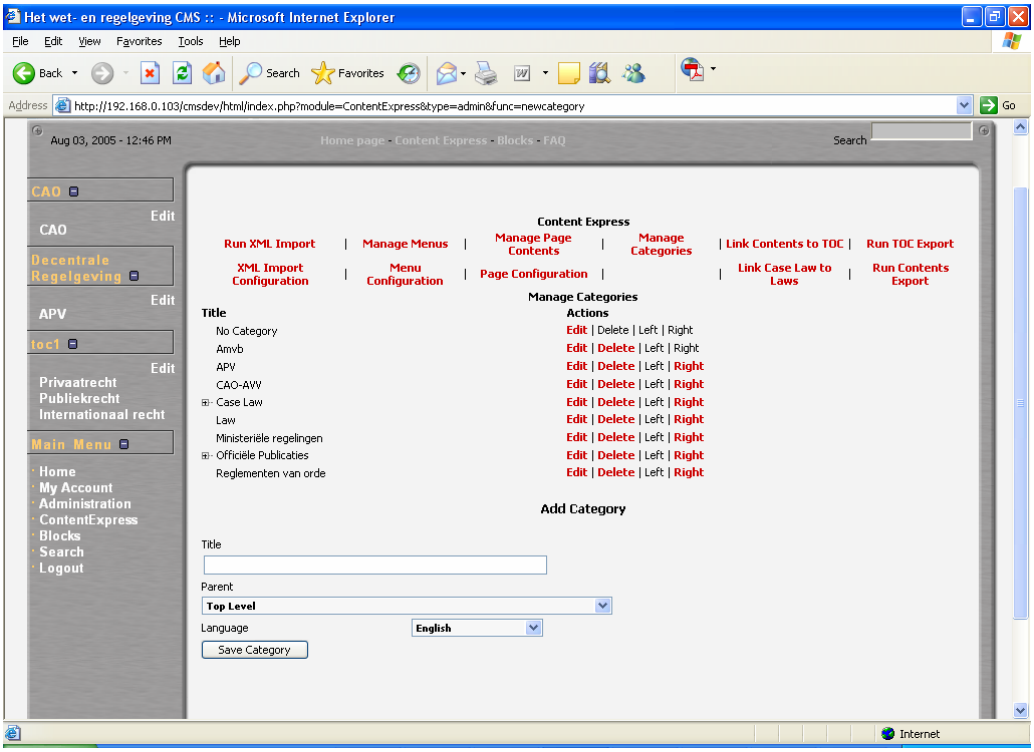
1. Navigate to "**ContentExpress**" from the main menu.



2. Navigate to "**Manage Categories**".



3. The main category management screen looks like:



4. To create a category, fill in these fields in the **Add Category** form:
    - **Title**: The category name.
    - **Parent**: Choose "**Top Level**" for top level menus. The hierarchical sub categories are created during the import process.
    - **Language**: Use English as default language.

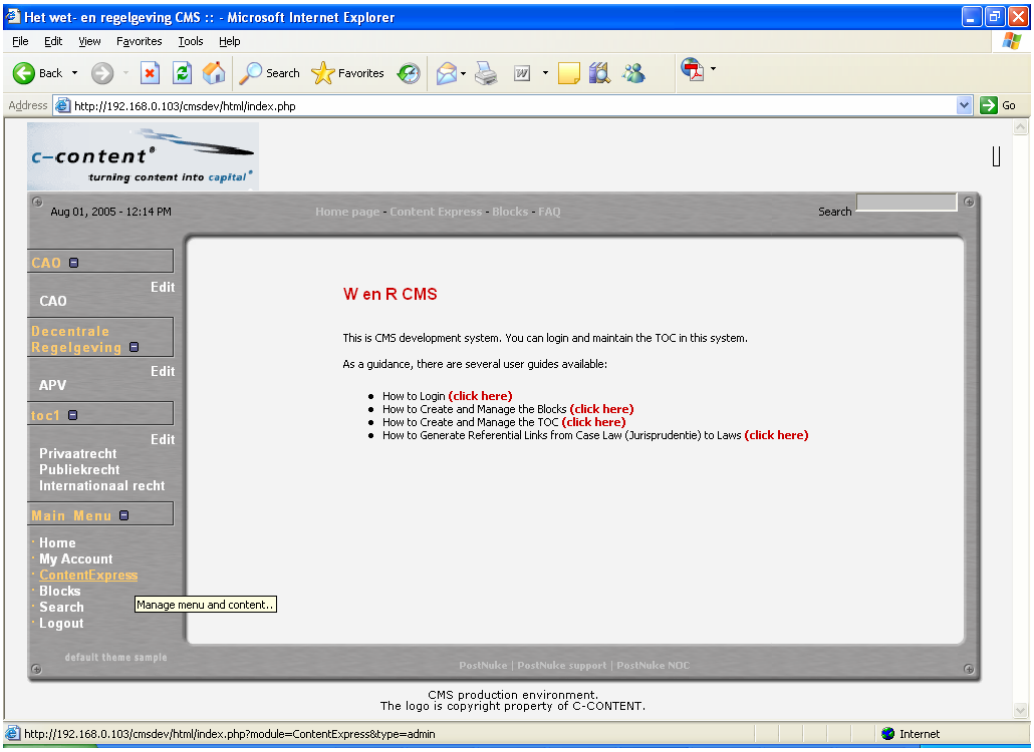
    Click the "**Save Category**" button to save your new menu.
5. You can **Edit** the category, and change the entries of the category (the same fields as if you create a new category).
6. You can **Delete** a category, if it is unused.
7. You can move the position of a category in the hierarchy by moving it:
    - Left: Move a category to be a parent level (one level above).
    - Right: Move a category to be a child level (one level below).

Configure and execute import process

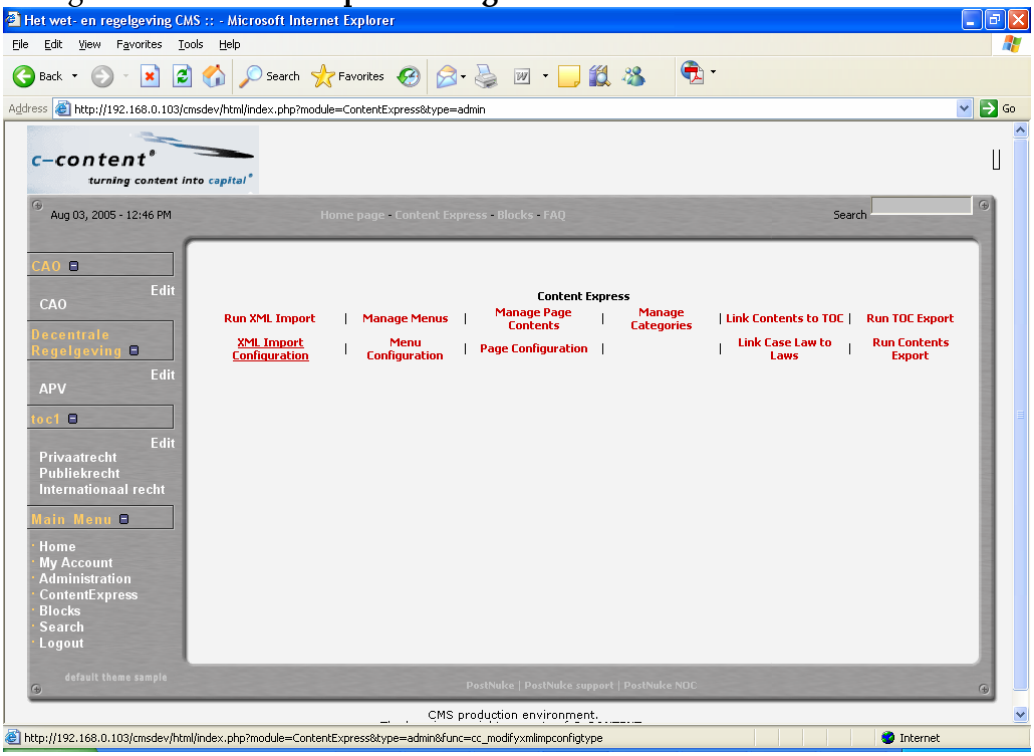
The import process consists of two activities, the configuration and the execution.

To configure the import process, the steps are:

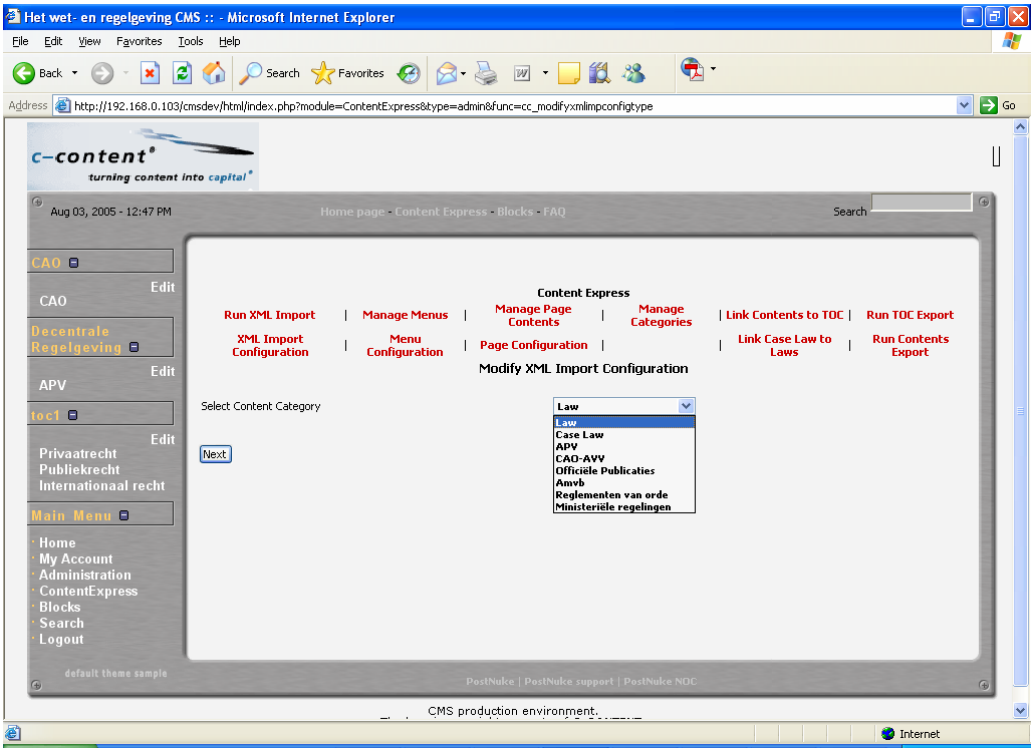
1. Navigate to "**ContentExpress**" from the main menu.



2. Navigate to the "**XML Import Configuration**".

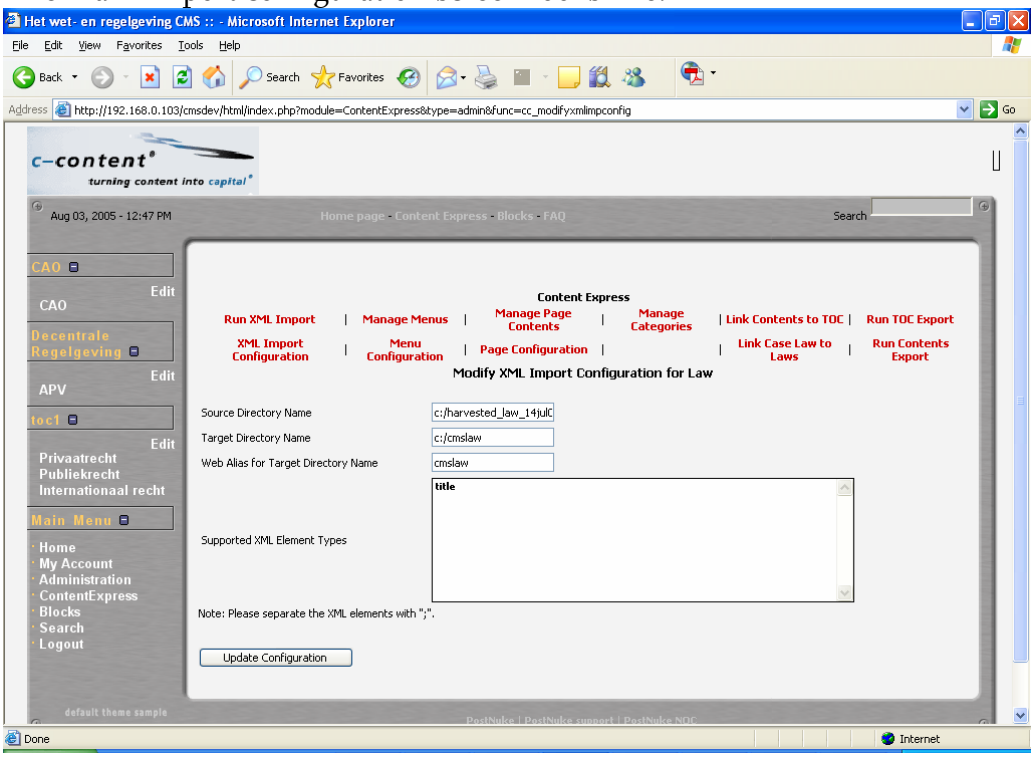


3. In the following screen, select the content category that will be configured. Then click "**Next**" button.



The category selection is based on the top level categories that created in the "**Create Category**" step. For new imported content category, you need to create the category before configure the import.

4. The main import configuration screen looks like:



5. The configurable import parameters are:
    - **Source Directory Name**: The location of the import source files.

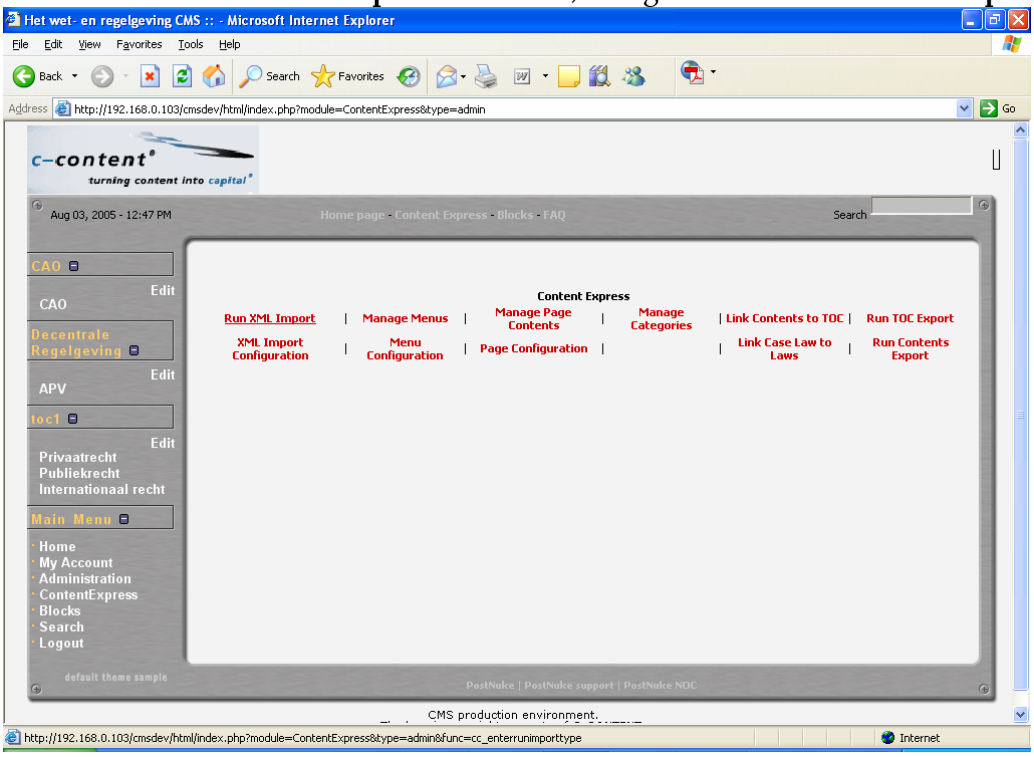
- **Target Directory Name**: The location of CMS files storage. We suggest creating separate directories for different content categories.
- **Web Alias for Target Directory Name**: The virtual address to be used in the web server. The same name should be defined in the web server configuration setting. This alias is not necessary if the content is in XML format, but the XSL should be developed to display the content.
- **Supported XML Element Types**: The XML element names that should be recognized during the import process.

Click the "**Update Configuration**" button to save your settings.

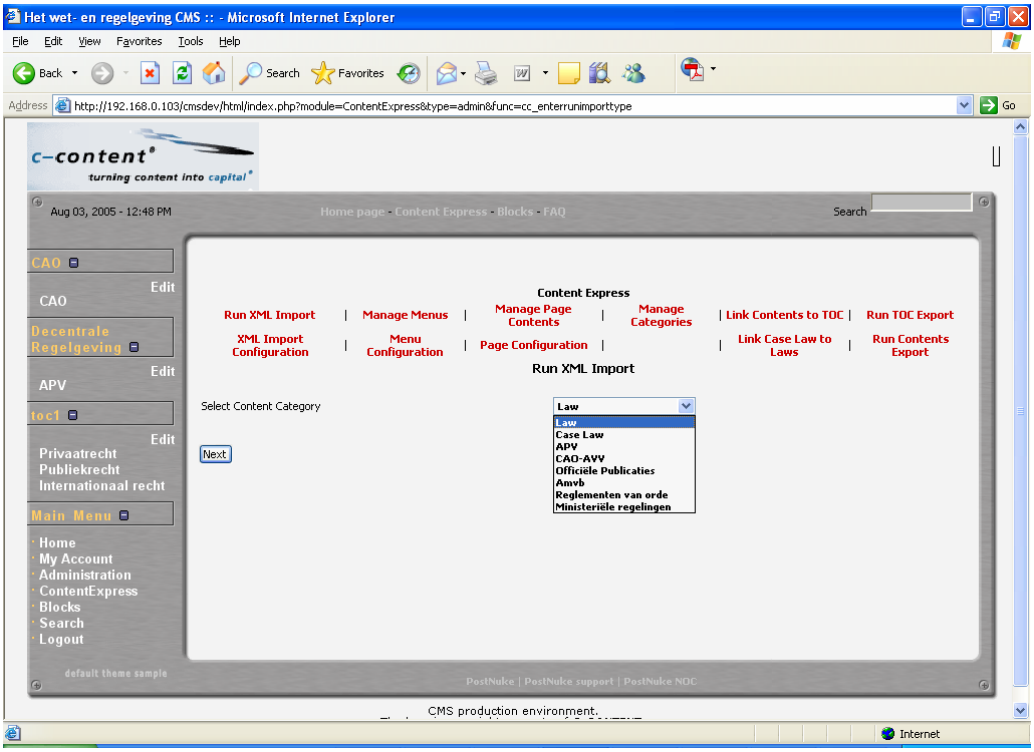
You can repeat the above steps for all content categories.

To execute the import process, the steps are:

1. From the main "**ContentExpress**" window, navigate to the "**Run XML Import**".

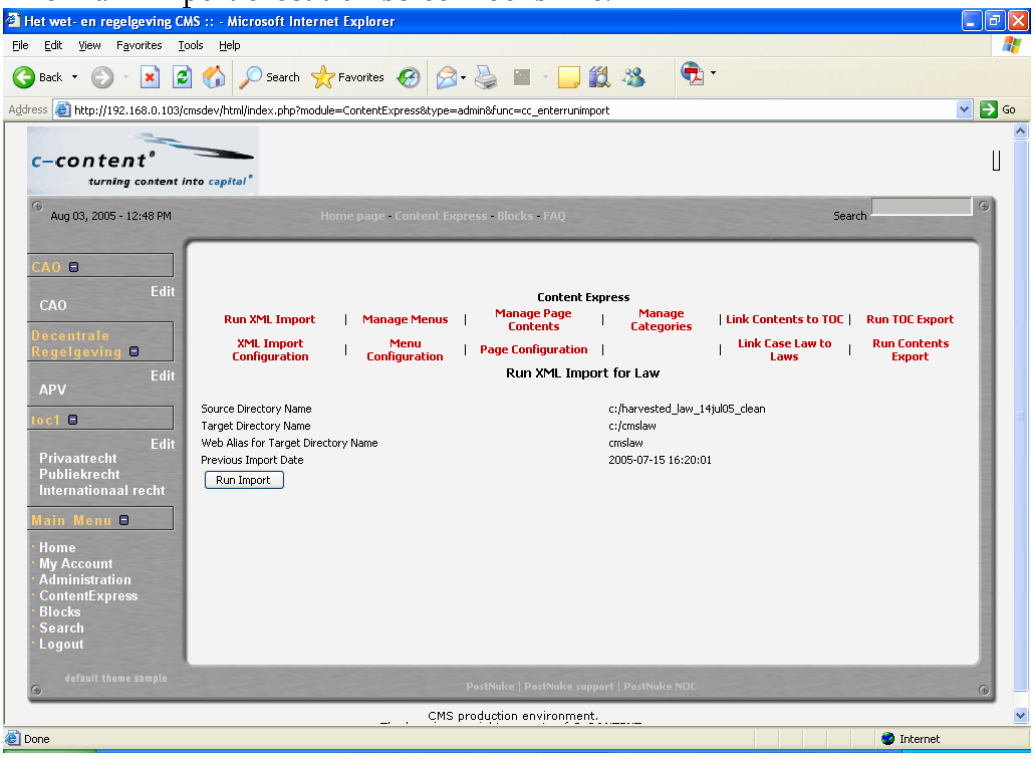


2. In the following screen, select the content category that will be imported. Then click "**Next**" button.



The category selection is based on the top level categories that created in the "**Create Category**" step. For new imported content category, you need to create the category before execute the import.

3. The main import execution screen looks like:



4. The information appear in this screen are the result of the import configuration. If the screen is empty then you need to check the configuration step for the appropriate content category.

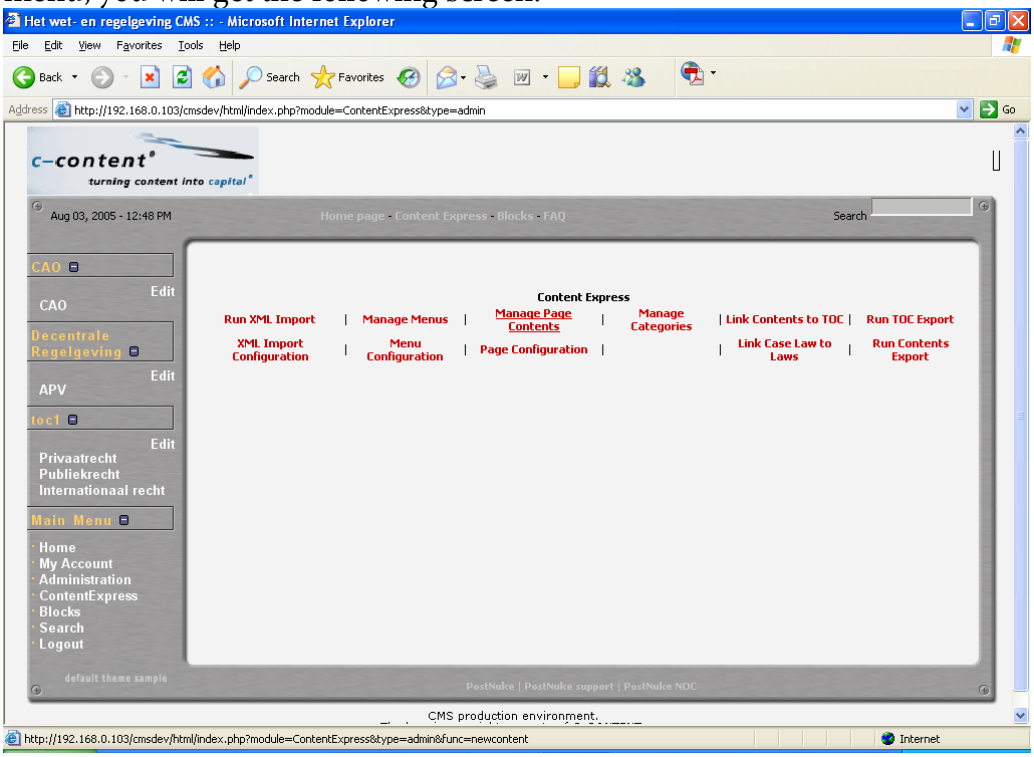
The only additional information is the "**Previous Import Date**", indicating the last timestamp of the import execution.
If you are sure with the parameters, click the "**Run Import**" button to execute the import process.

You can repeat the above steps to import all content categories.

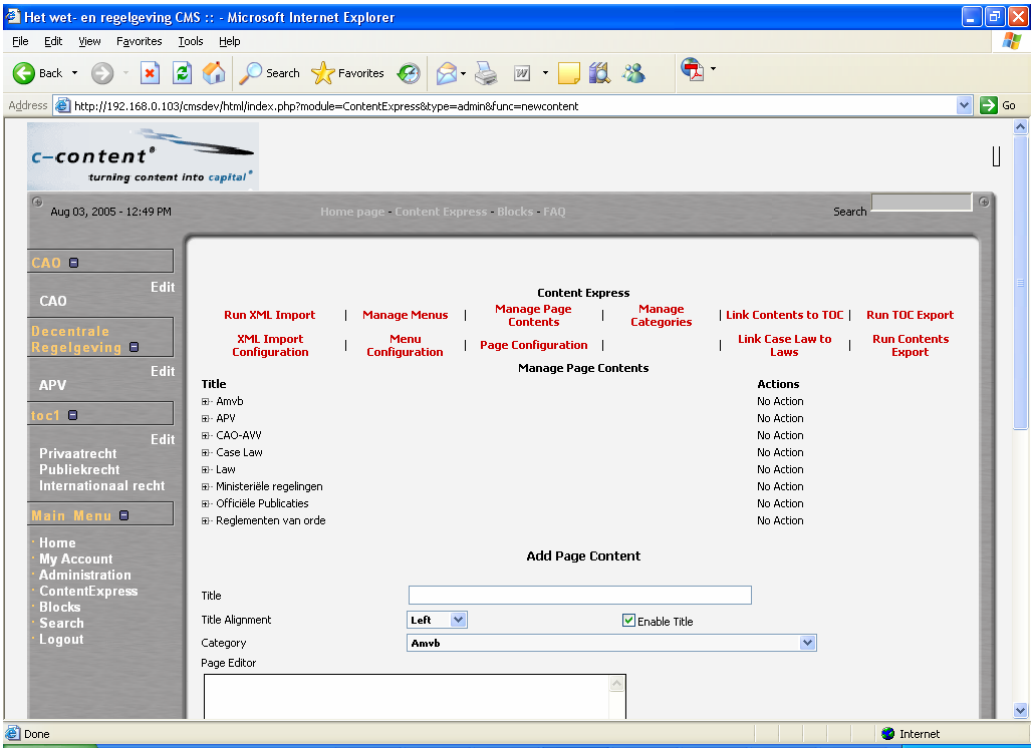
Generate referential links from case laws (jurisprudentie) to appropriate laws

Currently the CMS contains both the laws and case laws. The steps to create referential links from the case laws to appropriate laws are explained below.

1. After login to the CMS and navigate to the "**ContentExpress**" from the main menu, you will get the following screen.

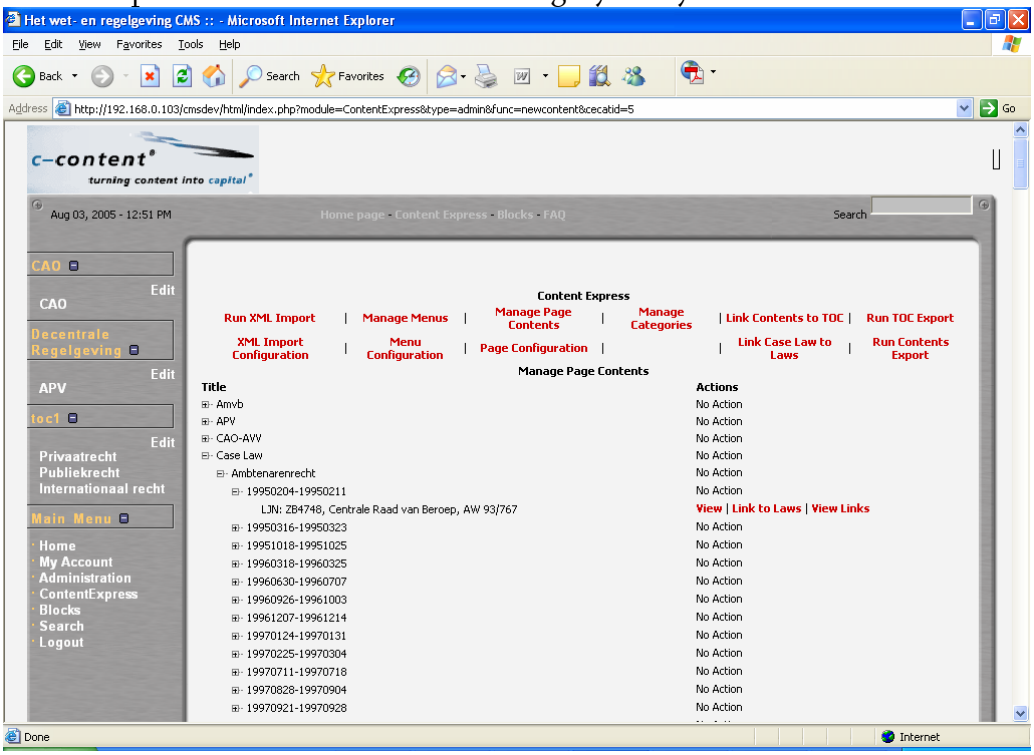


2. Navigate to "**Manage Page Contents**" to view the list of the contents (both laws and case laws).

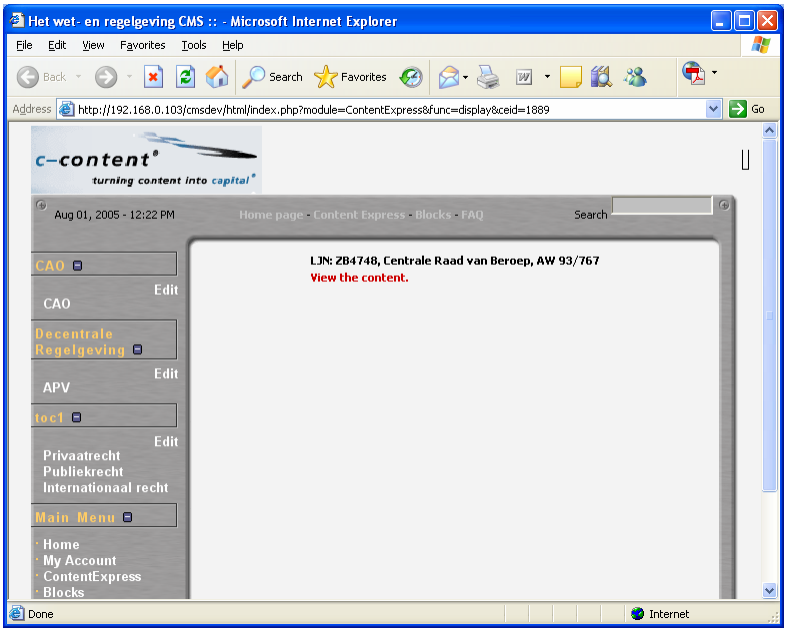


The contents are grouped based on content categories. The Case Law group is divided into several categories and then into several date periods based on the date of the case laws.

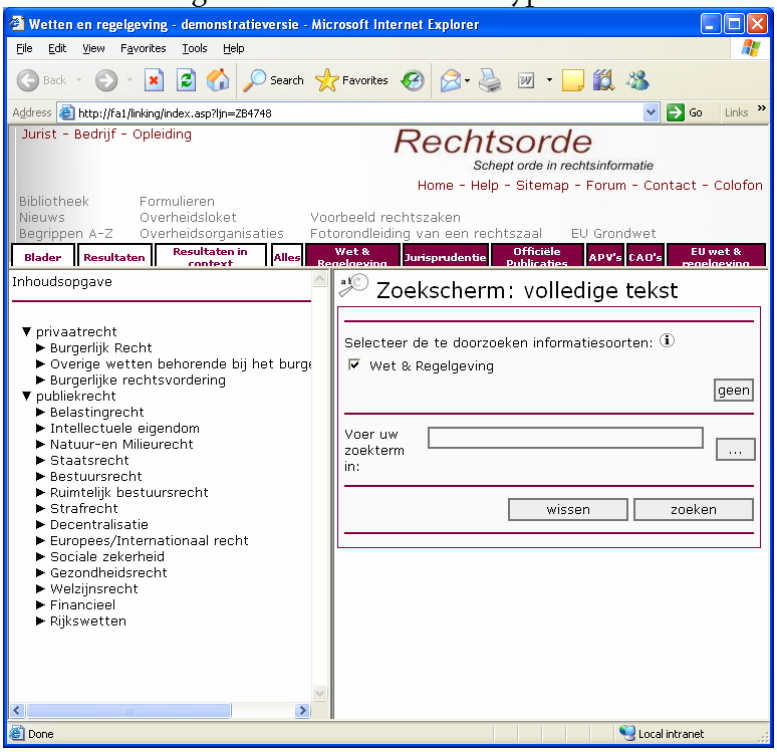
3. To view the individual case law, expand the hierarchy of the Case Law until you find the specific case laws under the category that you desire.



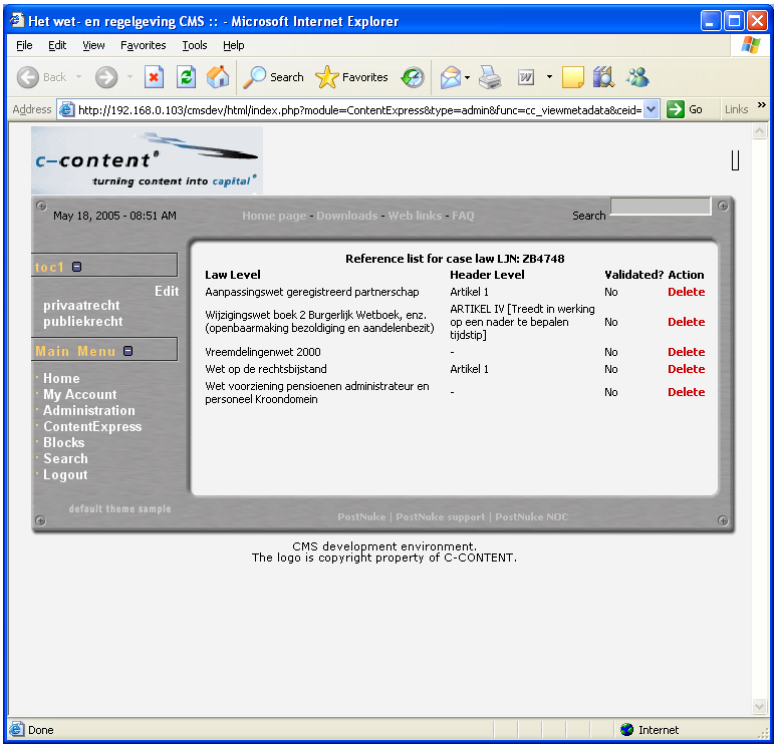
4. In this situation, you can have three possible activities:
   (1) View the case law by navigating to "**View**" hyperlink.



(2) Open the linking application to find the laws that being referred by a case law. Navigate to "**Link to Laws**" hyperlink to do this.

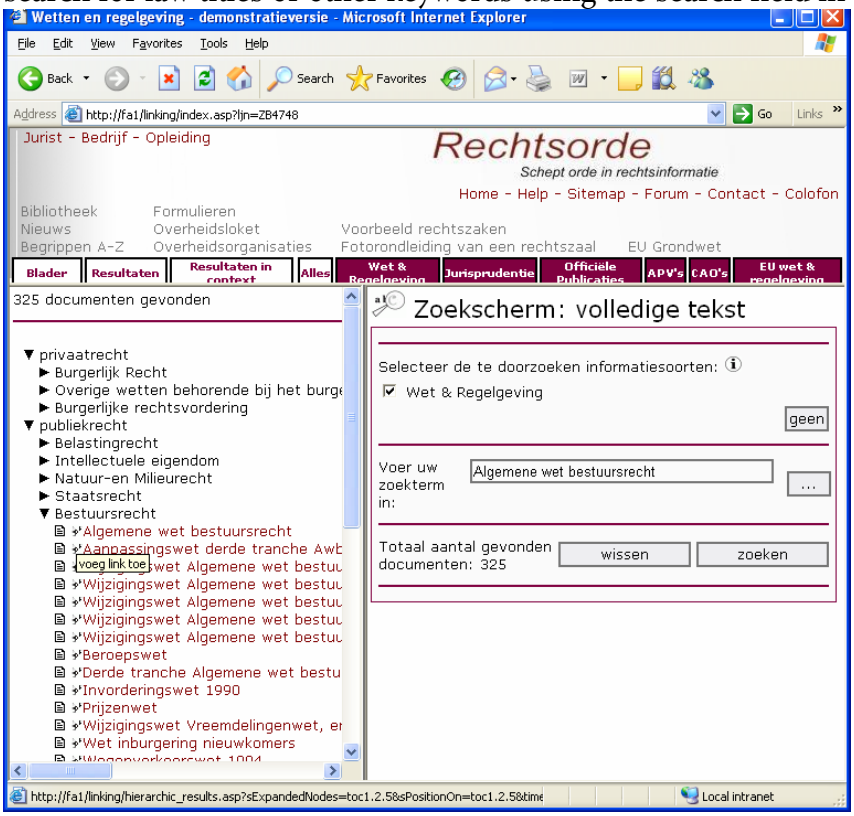


(3) View the existing referential links that you already created before or that suggested by the system, by navigating to "**View Links**" hyperlink. If the referential links are generated from the linking application, the "**Validated**" column will be "**Yes**"; otherwise it will be "**No**".

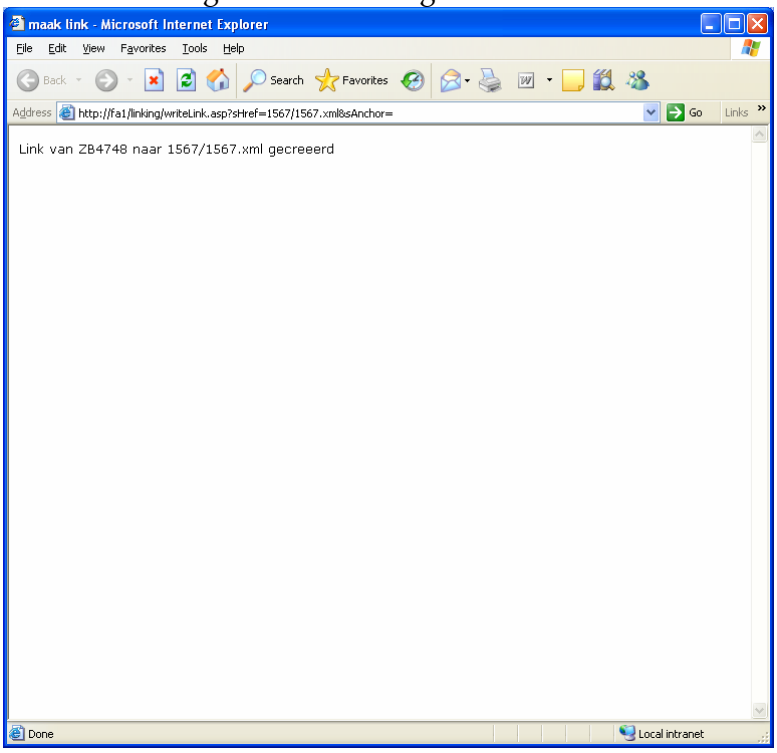


All these activities will be opened in new web browser window.

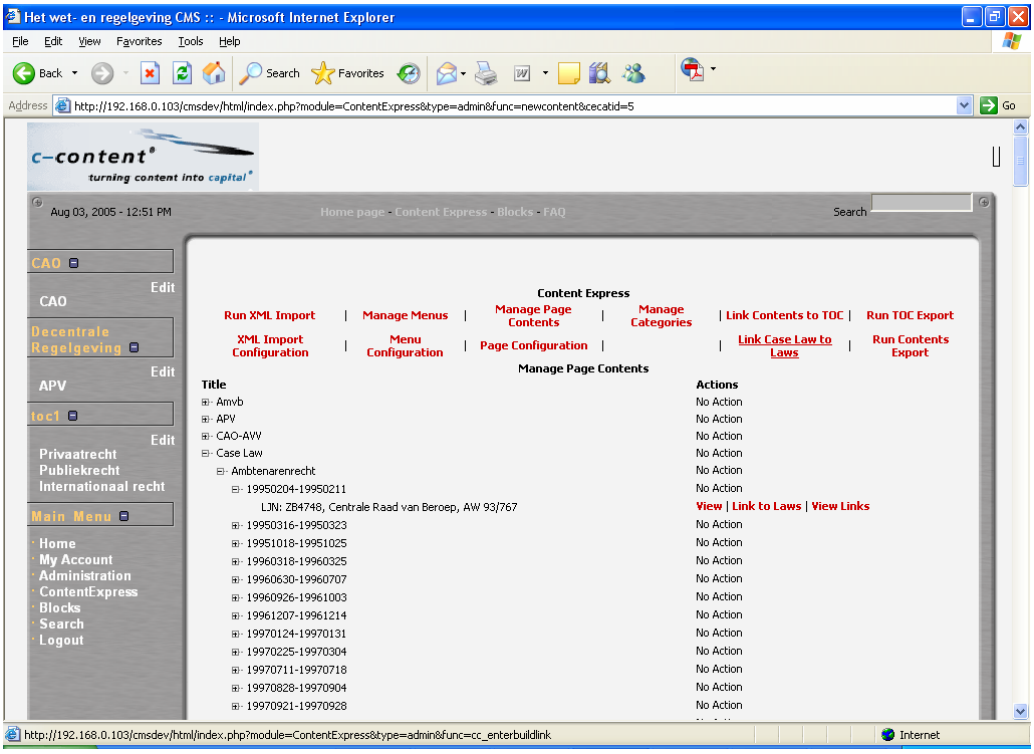
5. In the linking application window, you can browse through the TOC in the left side to find the law or law article that is referred by the case law. You can also search for law titles or other keywords using the search field in the right side.



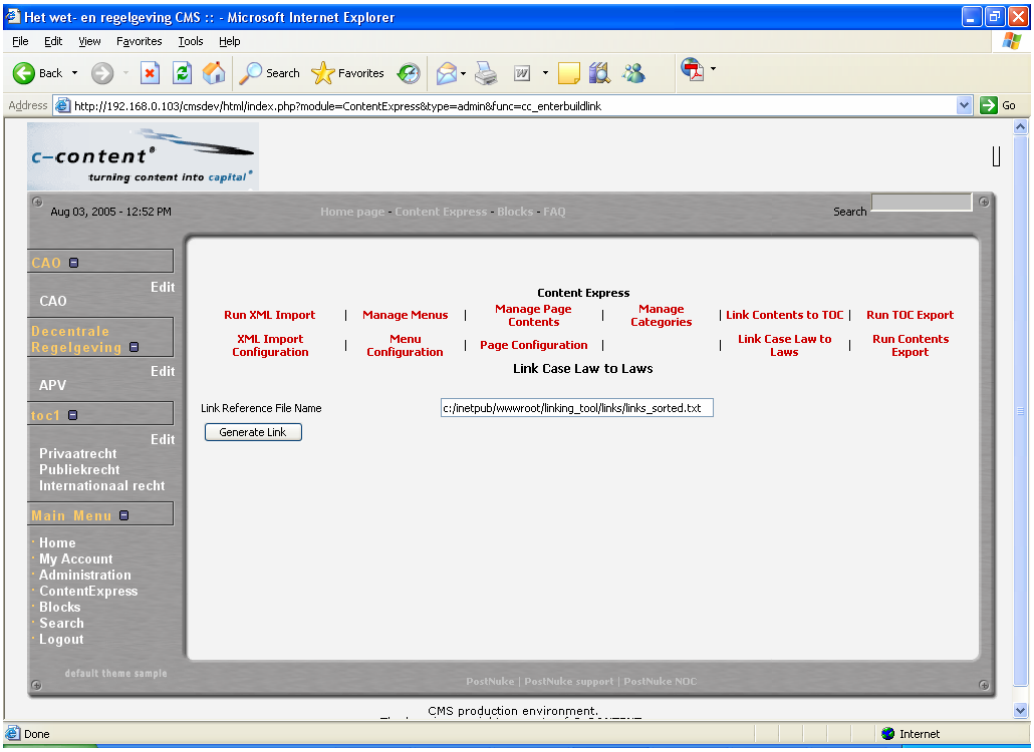
You can generate the link by clicking the icon beside the law or article in the TOC. You will get this following window as the notification.



6. After using the linking application, you can save the links that you create in the CMS. From the main menu in the ContentExpress in the CMS, navigate to the "**Link Case Law to Laws**" menu.

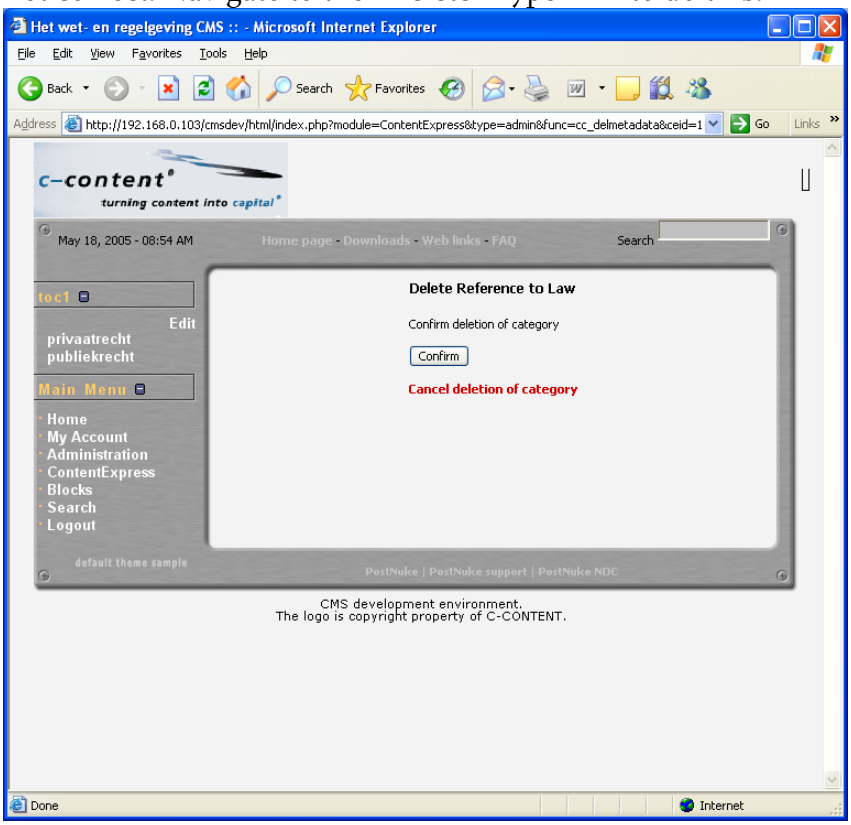


You need to key in the filename that used by the linking application to store the links information **(should be a default file)**.



Click the "**Generate Link**" button to save the links in the CMS. You can view the result in the list of the referred laws (step no. 4.(3)).

7. From the list of the referred laws, you can also delete a link if you think it is not correct. Navigate to the "**Delete**" hyperlink to do this.



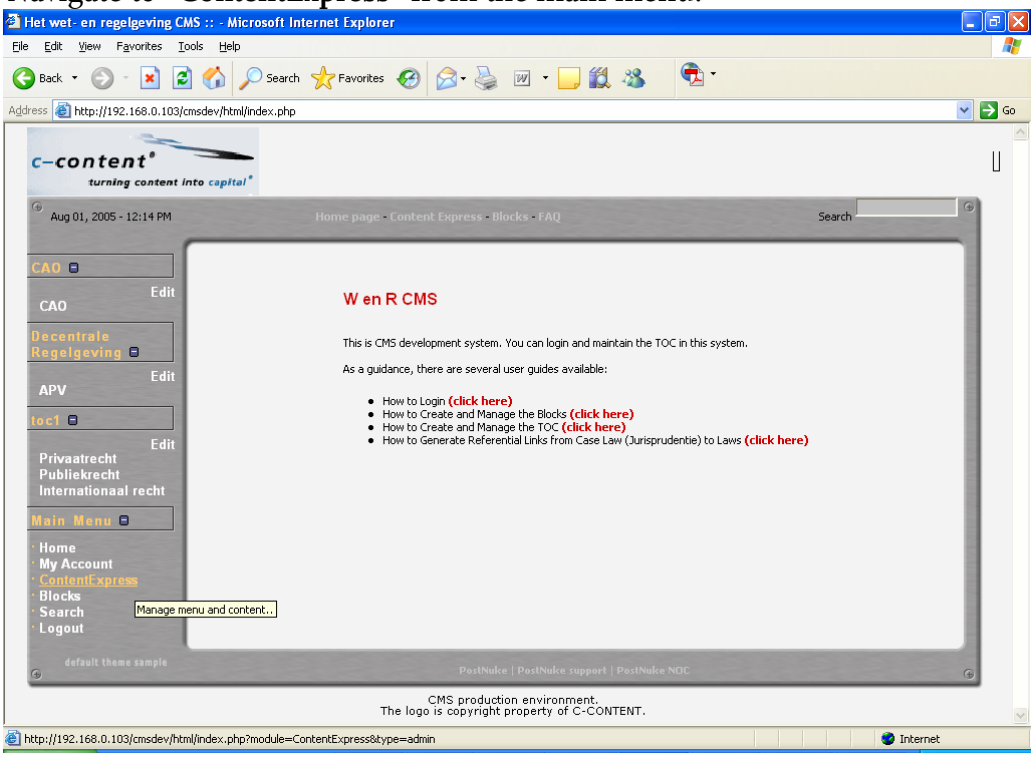
You need to give confirmation before the deletion is really executed.

Generate links between contents and the TOC

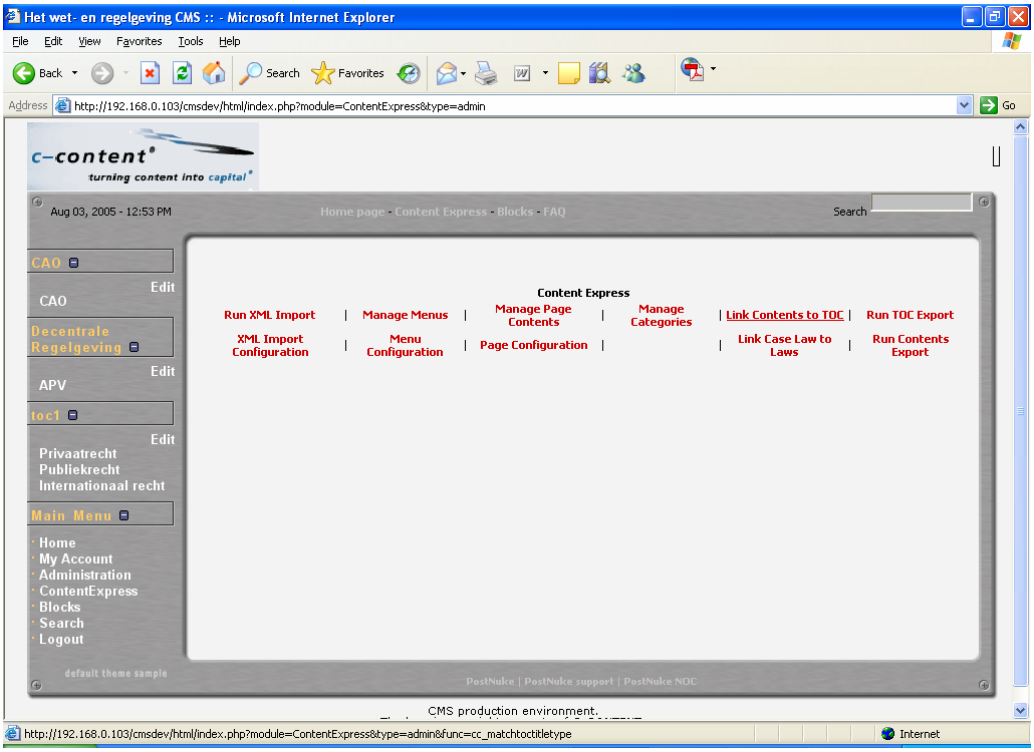
During the import process the imported contents are linked to the TOC. It is also possible that the TOC is modified without any import execution, and the links between the contents and the TOC need to be updated.

To update the links between the contents and the TOC, the steps are:

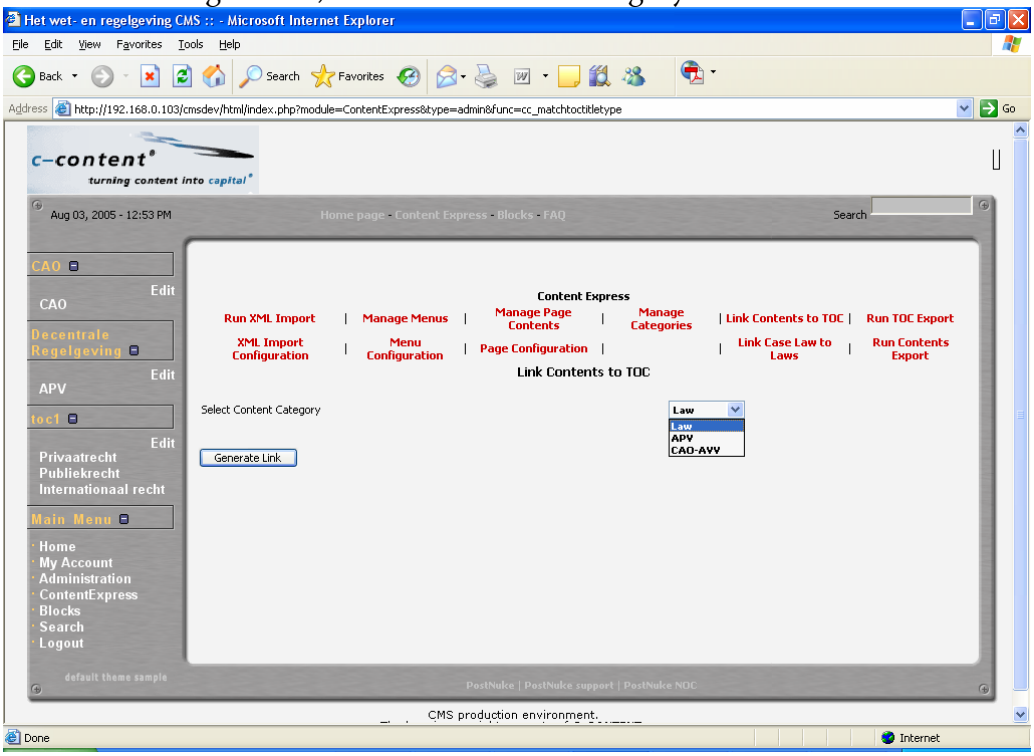
1. Navigate to "**ContentExpress**" from the main menu.



2. Navigate to "**Link Contents to TOC**".



3. In the following screen, select the content category that has link with the TOC.



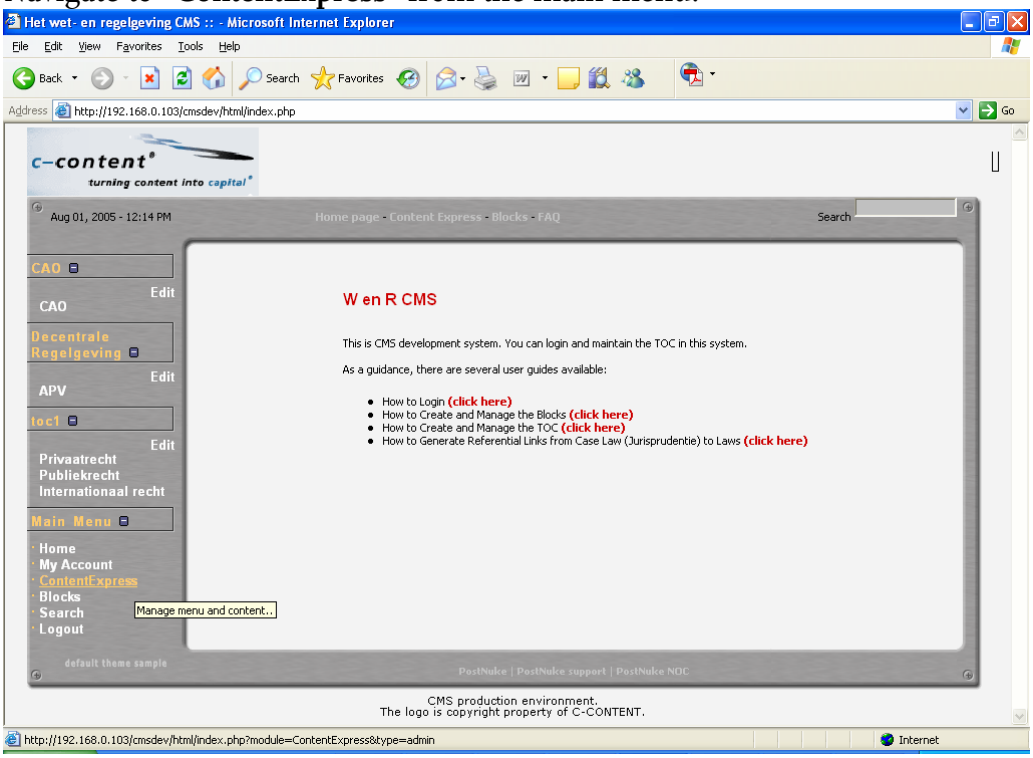
4. After the selection, click the "**Generate Link**" button to execute the process.

Execute export process

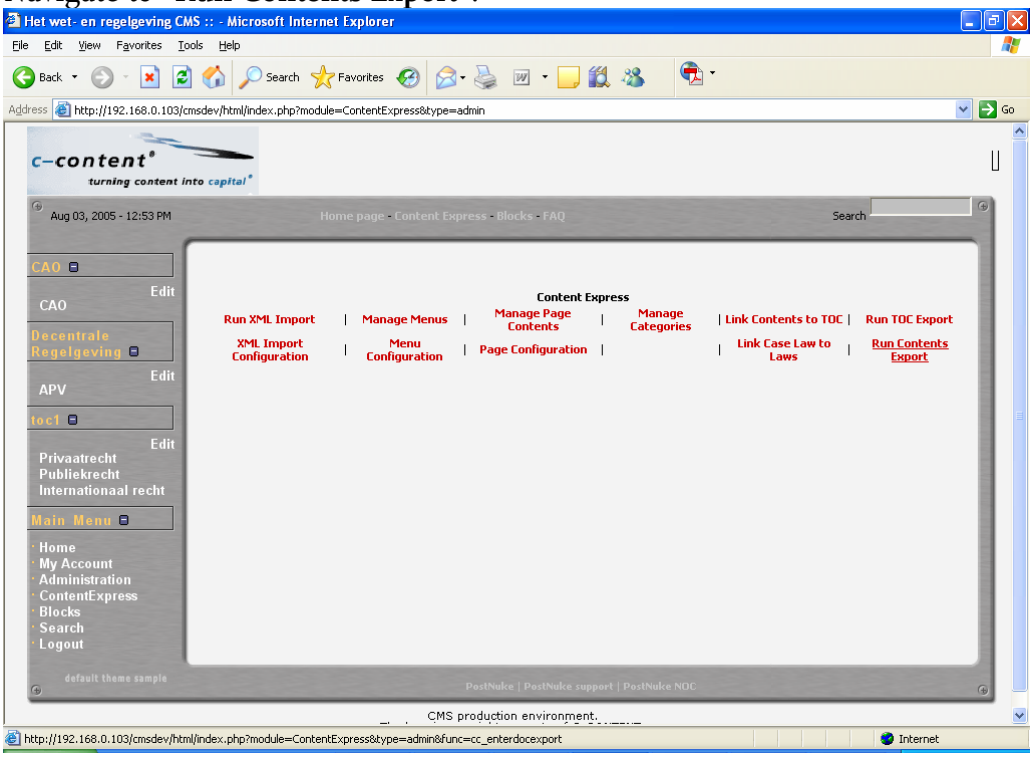
The export process consists of two execution steps, (1) exporting the contents and (2) export the TOC as XML file.

To export the contents, the steps are:

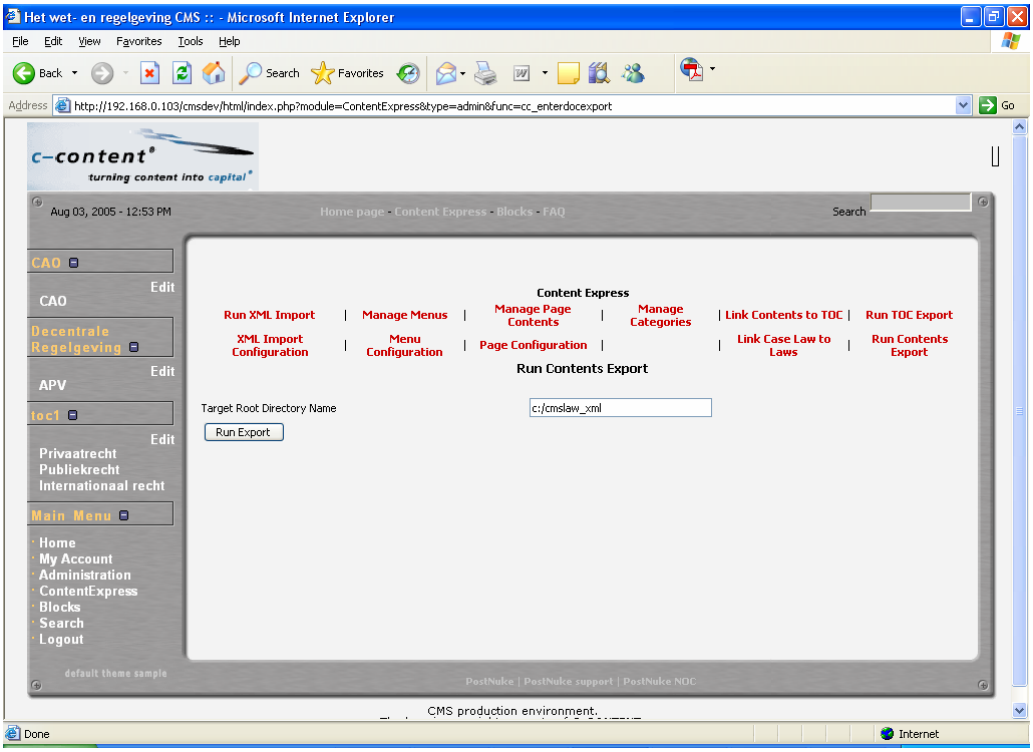
1. Navigate to "**ContentExpress**" from the main menu.



2. Navigate to "**Run Contents Export**".



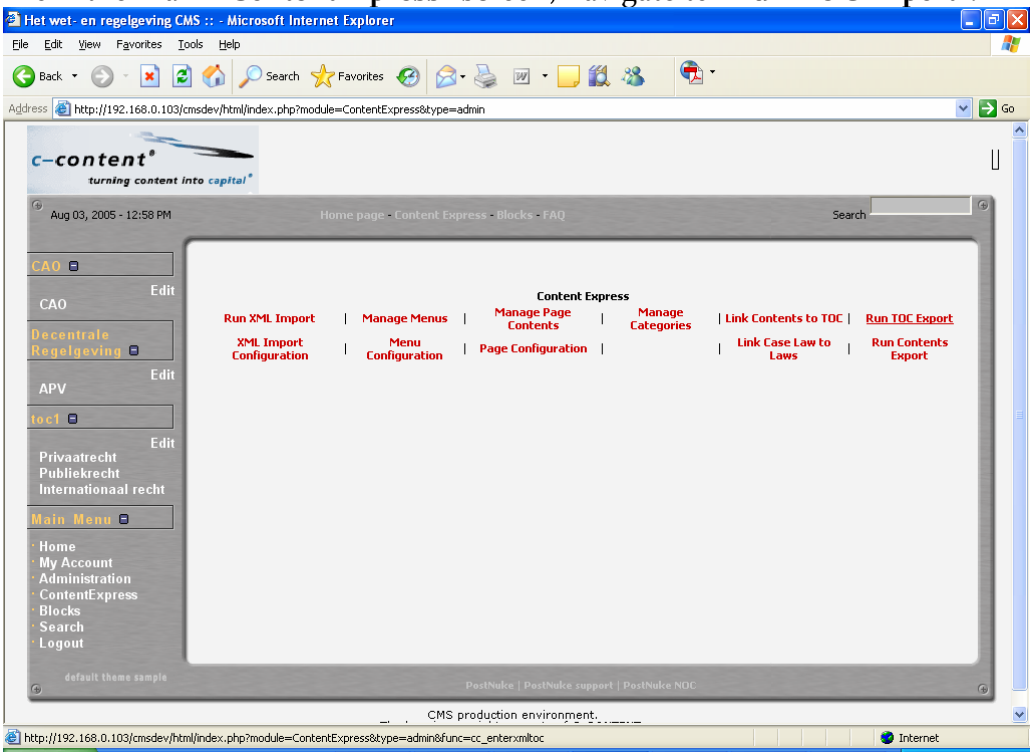
3. You will get the following screen:



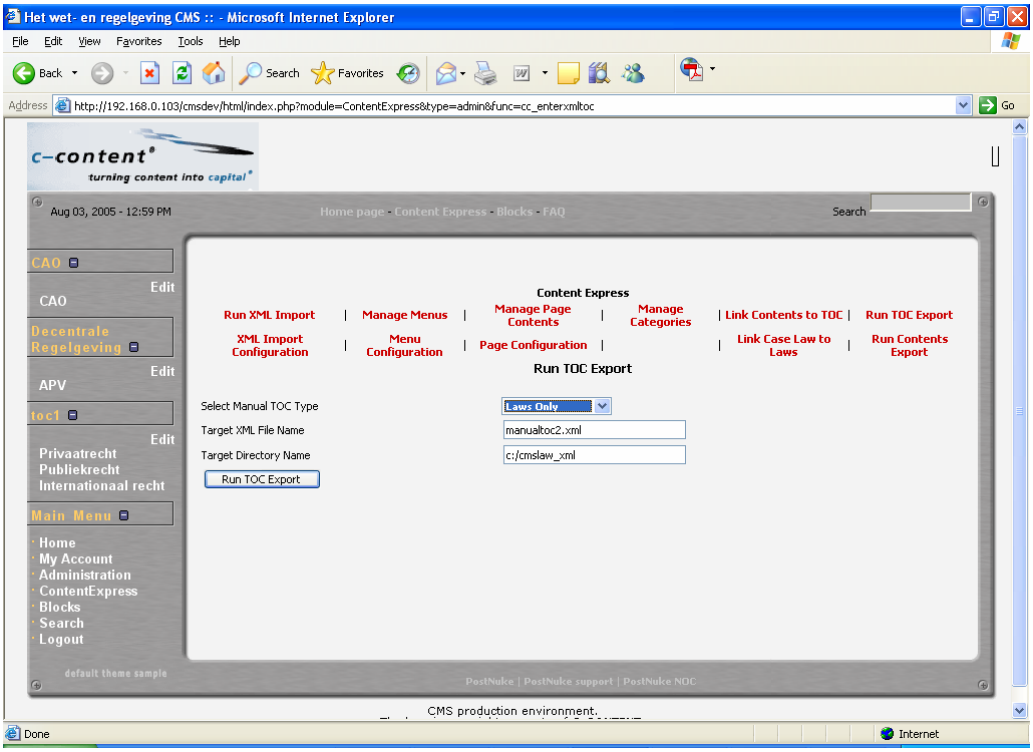
There is one export parameter, which is the "**Target Root Directory Name**". This is the directory where the export process writes the exported files.
After specifying the directory name, click the "**Run Export**" button to execute the contents export.

To export the TOC, the steps are:

1. From the main "**ContentExpress**" screen, navigate to "**Run TOC Export**".



2. The main TOC export screen looks like:



The parameters of the TOC export process are:

- **Manual TOC Type**: Option to export TOC without reference contents ("**Laws Only**") or including the reference contents ("**Laws-Case Laws**").
- **Target XML File Name**: The name of the generated XML file.
- **Target Directory Name**: The directory to store the generated XML file. Normally this is the same as the target directory of the contents export.

After specifying the parameters, click the "**Run TOC Export**" button to execute the TOC export.

Appendix C – Legend for WebML Diagrams

The WebML diagrams in Section 3.4 use the following legends.

Page and Page Area

 = Page Area

 = Page

Page Content Units

 = Hierarchical List

 = Flat List

 = Entry Form

 = Individual Data View

Operation Units

 = User Login

 = User Logout

 = Data Creation

 = Data Modification

 = Data Deletion

 = Non-database Operation

Arrows

 = Page Navigation

 = Operation Success

 = Operation Fail