


# **Roxen CMS 5.4**

Administrator Manual

 Roxen Internet Software AB  
© 2011 Roxen Internet Software AB.  
All rights reserved.

Under the copyright laws, this document may not be copied, in whole or in part, without the written consent of Roxen Internet Software.

Roxen Internet Software  
Box 449  
SE-581 05 Linköping  
Sweden  
[www.roxen.com](http://www.roxen.com)

Your rights to the software are governed by the accompanying software license agreement.

Every effort has been made to ensure that the information in this document is accurate. Roxen Internet Software is not responsible for printing or clerical errors.

Other company and product names mentioned herein are trademarks of their respective companies. Roxen Internet Software assumes no responsibility with regard to the performance or use of these products.

# Contents

---

<b>1</b>	<b>Getting Started</b>	<b>7</b>
1.1	WebDAV and FTP	7
1.2	Import & Export	7
1.2.1	Exporting data	8
1.2.2	Importing data	8
1.3	Jobs	9
1.4	File Types	9
<b>2</b>	<b>The Repository</b>	<b>11</b>
2.1	The cvsroot/site repository directory	11
2.2	New or changed files	11
2.3	New or changed directories	12
<b>3</b>	<b>Languages</b>	<b>13</b>
<b>4</b>	<b>Links to External Files</b>	<b>14</b>
4.1	Shared directory	14
4.2	Create a link	14
<b>5</b>	<b>SmartSearch</b>	<b>15</b>
5.1	Background	15
5.2	Performance optimizations	15
<b>6</b>	<b>Application Launcher</b>	<b>17</b>
6.1	Windows	17
6.2	UNIX	18
6.3	Installing the Application Launcher	18
6.4	Local Editors	19
6.5	.roxen_launcher.rc	19
6.6	Editor Profiles	19
<b>7</b>	<b>Access Control</b>	<b>23</b>
7.1	Zone Navigation	23
7.2	Users and Groups	24
7.2.1	Users	26

7.2.2	Groups	26
7.3	Protection Points	26
7.3.1	Protection Points in External Modules	27
7.4	Adding Protection Points	27
7.4.1	Protection Points in File Structure	27
7.4.2	Protection Points in RXML	27
7.5	Permissions	27
7.5.1	Setting Permissions	28
7.6	Protection Classes	29
7.6.1	Resources	29
7.7	Predefined Protection Classes	29
7.7.1	Module specific protection classes	29
7.7.2	Default protection classes	30
7.8	Zones	32
<b>8</b>	<b>Persistent Cache in SiteBuilder</b>	<b>33</b>
8.1	Setting Up	34
8.2	Enable/disable persistent disk cache	34
8.3	Crawler configuration	34
8.4	Prefetch restrictions	35
8.5	Top-level caching	35
8.6	Force access as Everyone	35
8.7	First time setup	36
8.8	RXML Compilation	36
8.9	Cache Monitoring and Performance Analysis	38
8.9.1	Cache crawler status	38
8.9.2	Logging of \$cache-status	38
8.9.3	Resolve path	39
8.9.4	XSLT and RXML profiling in Content Editor	39
8.10	Multiserver Configuration	40
<b>9</b>	<b>Database Maintenance</b>	<b>41</b>
<b>10</b>	<b>Replication</b>	<b>42</b>
10.1	Replication Methods	42
10.2	Replication Setup	44
10.3	Setting up the server	44
10.3.1	Setting up the client	46
10.3.2	Setting up the replication directory	46

<b>11</b>	<b>Workflow</b>	<b>48</b>
11.1	Concepts	48
11.2	Definition files	48
11.3	User interface	50
11.4	Workflow XML syntax	50
11.4.1	Sample workflow definition	50
11.4.2	Basic structure	51
11.4.3	Attributes	51
11.5	Important notes on implementation	53
<b>12</b>	<b>Link Management</b>	<b>54</b>
12.1	Overview	54
12.1.1	Introduction to Link Management	54
12.1.2	File UUIDs	54
12.1.3	Link format	54
12.1.4	Compatibility	55
12.2	User interface	55
12.2.1	Administrator settings	56
12.2.2	Reassigning Unique Identifiers	56
12.2.3	Export and Import wizards	56
12.2.4	Administrator interface actions	56
12.3	Migration strategy	57
12.3.1	Step-by-step guide	57
12.4	Implementation notes	57
<b>13</b>	<b>Multiserver</b>	<b>59</b>
13.1	Acceleration Frontends	59
13.1.1	Multiple Edit Servers (MES)	59
13.1.2	Acceleration Server Setup	60
13.1.3	Backend Server	60
13.1.4	Acceleration Server	60
13.2	Argument Cache	60
13.2.1	Replicated argument cache	61
13.2.2	Configuration	61
13.3	Multi Edit Server	62
13.3.1	Overview of the design	62
13.3.2	Notes about usage	63
13.4	Multi Edit Server Installation	63
13.4.1	Base installation and configuration	63
13.4.2	Editor Backend configuration	64
13.4.3	Editor Frontend configuration	64
13.4.4	Access Control (AC) configuration	64
13.4.5	Search configuration	65
13.5	Test Server Setup	65

13.5.1	Live to Test copy	65
13.5.2	Test to Live copy	66

# 1 Getting Started

---

SiteBuilder is installed by creating a *CMS Advanced site*.

The second step when installing SiteBuilder is to create the users that are to have access to the Content Editor. If the users already are in the user database of the operating system you can do this by enabling and configuring the AC: OS user import module.

The third step is to make sure that the users can edit files with the editor of their choice. This means using Roxen Application Launcher, which will enable the users to use a local editor running on their computer. On the server end some sort of file sharing between the web server and the users' computers need to be installed and SiteBuilder configured to use it. See the Application Launcher page for detailed information on how to do this.

All files used by SiteBuilder are stored in a separate directory structure. When moving a SiteBuilder site between different servers all you have to do is to configure the SiteBuilder storage in the SiteBuilder module to point at this directory. The directory structure can be copied to another location, no absolute paths are used.

A SiteBuilder server may spawn several sites. Work areas should be mounted in different sites since they might contain absolute paths and thus not work when mounted anywhere but "/". To be able to edit through FTP you have to use another site, since all the modules of a normal SiteBuilder site will otherwise show up in directory listings.

When using several sites the SiteBuilder work area file system and AC: User database modules must be enabled, and configured to connect to the appropriate SiteBuilder. The SiteBuilder, SiteBuilder content manager and any Access Control modules but AC: User database should not be enabled in such a virtual server. Other modules, like the SiteBuilder tags module can be enabled if their functionality is wanted.

## 1.1 WebDAV and FTP

To make your SiteBuilder site available through WebDAV or FTP, you need to create a CMS WebDAV/FTP edit site. Configure a port with the http or https protocols to use WebDAV, and/or a port with the ftp protocol to enable FTP editing.

To make FTP editing work with logged in users, you need the AC: User Database module. It must be configured to connect to the Access Control in the right SiteBuilder server.

The SiteBuilder Work area Filesystem module should have its Download mode variable configured to All work areas. The SiteBuilder variable should be configured to the appropriate SiteBuilder site. With this configuration all work areas will be available for edit in different subdirectories.

## 1.2 Import & Export

As a site administrator you can export a subtree of your edit area to the server hard disk. The exported data can later be imported in another location and/or a different site. The exported data will preserve metadata for each file but no revision history.

If users of these commands don't have access to the server hard disk they can also use an option to download and upload the data as a ZIP archive.

Remember that some metadata may be specific to a given site. E.g., if an XML file identifies a XSLT template by name and that template is not exported and not defined in the import site, the XML file will not render correctly. Likewise, workflow metadata and content types may need special attention.

**Note!**

If the Export and Import commands described below aren't available, make sure you have enabled access to the Site Export/Import protection point in the Access Control.

**Note!**

ZIP export/import requires the server machine to have a command-line ZIP utility installed. On Unix-based systems the default commands are zip and unzip; Windows-based installations have no default setting. To customize the commands, go to the Administration interface and navigate to Sites -> (your site) -> CMS: Content Editor -> Settings tab and change the settings named Create ZIP command and Expand ZIP command. Leaving the settings empty will disable ZIP export/import capabilities.

### 1.2.1 Exporting data

To export a directory, select Export... from the File popup menu in the Files for the directory where you want to start. All files and directories enclosed in the starting directory will be included. Before the export is performed you are asked to name the collection of files that you want to share. You will later find a corresponding directory called site\_name.sb/transport/collection\_name/ on disk.

In addition to storing the files on the server hard disk, checking the Download to local computer in ZIP format option will send a compressed version to your computer. This can later be used with the Import command in a different directory or a different Roxen CMS server. Previous ZIP exports are also listed in the Export dialog for repeated downloading.

**Note!**

If the export process takes a long time your web browser may time out before all work is completed. This will not affect the server but you will not get an alert message when the export is done. The same message is added to the server log so check it to be sure the export is finished.

### 1.2.2 Importing data

If you want to import into a different site than the one you exported from you must start by copying the data in the path above to the corresponding location in the new site. Next, select the Import... command from the File popup menu and select the matching collection name from the menu.

If access to server hard disk is not possible and the data you want to import exists in a ZIP archive, you can upload this archive via the browser. It will then be placed in the transport directory and subsequently unpacked. To use this feature, click the Upload archive in ZIP format... link, select the ZIP file on your local computer, enter an import name and click Upload & Refresh.

Prior to running the command you get a chance to preview pathnames for the files that will be created. At this point you can verify that your imported files will end up in the correct place.



The imported data will be placed in your edit area. Use the Commit command to publish it on the site.

## 1.3 Jobs

SiteBuilder can execute RXML tags such as AC and SB tags using a command based interface. The program is called sb-job and is located in roxen/server-\*/modules/sitebuilder/bin/sb-job. This is useful for batch processing, automated operations and general script based control from e.g. a unix prompt or cron.

The sb-job program takes the following arguments:

```
sb-job <SiteBuilder storage dir> [job file or "-" for stdin]...
```

A job file is a standard XML file that contains the RXML tags that will be executed by SiteBuilder. The job files are executed asynchronous (the program quits before the statements have been executed) as the Administrator User (typically AC id 2) by default. Jobs are put in a job queue in the SiteBuilder Storage directory and SiteBuilder need not be running when the sb-job program is run. When SiteBuilder is running, it is polling the job queue regularly and executes pending jobs automatically.

To debug jobs, it is useful to use the <debug werror="..."/> tag. It prints its output in the standard logs/debug/default.1 log file.

## 1.4 File Types

A SiteBuilder site handles different file types. The file types are presented to the user as a string describing the file type and an icon when doing directory listings. It is possible to use the same icons in file listings done in RXML, by using the <emit> tag.

File types have handlers that are plugins that handle upload, download and viewing of the file. It is important that each file type get the right handler, the text handler would for example not work very well with an image file. For new file types the default handler should be used.

The handlers are:

### **default**

For any file type that cannot be handled by other handlers. Only the basic SiteBuilder functionality will be available for files handled by the default handler.

### **image**

For handling image files.

### **html**

For handling HTML files.

### **text**

For handling any text file. It will be possible to edit with the Content manager.

### **menu**

For handling menu files, that should have the content-type sitebuilder/menu.

### **template**

For handling template files, that should have the content-type sitebuilder/template.

**external**

For handling links to external files, that should have the content-type sitebuilder/external.

File types are configured with the File types button under the Configuration tab. For each file type the following data can be configured:

**Content type**

The MIME content type of the file type. If the content type begins with roxen/ the file will not be handled by SiteBuilder but rather sent unmodified to Roxen WebServer. That means that Roxen WebServer's file extension modules will be used to handle the file. It also means that the correct file extension must be used.

**Alt Content Type for Editing**

You can specify a secondary content type that can be used for invoking an alternative editor.

Example usage: The "CMS Page Editor" file type normally invokes the built-in component editor, but sometimes it makes more sense to edit the file as an XML file since XML is the underlying format of those files. By setting this option to XML, the "Edit as XML" command will be available for "CMS Page Editor" files alongside the standard edit option.

**Name**

The name of the file type. Will be presented to the user.

**Image**

The image to display in the content manager for this file type.

**Handler**

Which handler to use for this file type.

**Download**

Whether it should be possible to download this file type or if it should only be handled through the interface in the content editor.

**Parse**

Whether this type of files should be run through the template system and the RXML parser. Will only have effect if the handler supports parsing.

**Extensions**

The file extensions of this file types. Is used to automatically determine an appropriate file type when uploading a new file. It is by no means necessary for files to have these extensions.

**Default File**

This is a path to either a file or directory. The file, or files, will be used as default files in the Edit new file wizard. It is necessary to have default files to be able to create new files of the file type from within SiteBuilder.

## 2 The Repository

---

A SiteBuilder site is stored using CVS. Basic knowledge about CVS is recommended when working with these files and directories. Explaining CVS is out of scope for this section - more information on CVS can for example be found at The GNU CVS site.

SiteBuilder normally handles all needed operations in these directories automatically. Apart from debugging, certain external operations are allowed within the repository (see below).

The SiteBuilder storage directory (typically ends with .sb) contains two CVS related subdirectories:

- `cvsroot` - this directory always contains the subdirectory `site` which is where the SiteBuilder CVS repository is located.
- `wa` - this work area directory always contains the subdirectory `view` which is where SiteBuilder keeps the checked-out versions of committed (published, non-hidden committed) files. The `wa` directory may also contain edit area subdirectories, with numerical names based on the Access Control id number for each user respectively.

### 2.1 The `cvsroot/site` repository directory

The contents of the repository (`cvsroot/site`) directory can be manipulated by external tools outside SiteBuilder, provided the following interface criteria are met. This is useful for example when copying versioned files (keeping version history etc.) between different SiteBuilders, archiving, resource sharing between edit servers etc.

- The repository directory can be manipulated externally. The `wa` directory is automatically adjusted by SiteBuilder accordingly.
- SiteBuilder only uses files ending with `,v`. Other files are currently ignored.
- SiteBuilder always assumes that `,v` files are structurally complete and correct. A `,v` should never exist in a broken or half-finished state. File changes must be atomic.
- SiteBuilder handles file and directory changes on-demand. It is up to the applications that SiteBuilder discovers changes in a timely manner:
  - The flag `EXTERNAL_REPOSITORY_UPDATES` will within a few seconds stat file dependencies through normal tags such as `<emit#dir>`, `<include>` etc. as long as these operations are not cached.
  - The `<sb-notify-change>` tag can be used to force SiteBuilder to check if a file or directory has been modified.

### 2.2 New or changed files

SiteBuilder will consider a file new or changed based on the following conditions:

- The corresponding file does not exist in `wa/view`.
- The `,v` file is newer than the corresponding file in `wa/view`.

## 2.3 New or changed directories

SiteBuilder will consider a directory new or changed based on the following conditions:

- The corresponding directory does not exist in `wa/view`.
- The directory metadata `__info,v` file is newer than the corresponding file in `wa/view`.
- SiteBuilder will automatically delete files and directories from `wa/view` if these have been deleted in the repository.
- SiteBuilder issues `file_changed`, `dir_changed_flat` and `dir_changed_recursive` hooks accordingly when discovering changes in the repository directory.
- To avoid excessive warnings in the debug log, the flag `EXTERNAL_REPOSITORY_UPDATES` can be used.
- The `EXTERNAL_REPOSITORY_UPDATES` flag does not currently work together with `hidden-commits`. `hidden-commits` occurs e.g. when a document that is within a workflow moves from one workflow stage to another and the target stage is a non published state.
- Applications must ensure that files changed externally are not at the same time manipulated by SiteBuilder itself.

# 3 Languages

---

For each file it is possible to set in the metadata what language it is in. This will affect which encoding it is supposed to have. The Languages button under Configure can be used to configure the languages that should be available for the site.

Code	Name	Encoding	Dictionary	Actions
	None	ISO-8859-1	en	
ar	Arabic	ISO-8859-6	ar	
da	Danish	ISO-8859-1	da	
de	German	ISO-8859-1	de	
en	English	ISO-8859-1	en	
es	Spanish	ISO-8859-1	es	
fi	Finnish	ISO-8859-1	fi	
fr	French	ISO-8859-1	fr	
it	Italian	ISO-8859-1	it	
ja	Japanese	UTF-8	ja	
nl	Dutch	ISO-8859-1	nl	
no	Norwegian	ISO-8859-1	no	
pl	Polish	ISO-8859-2	pl	
pt	Portugese	ISO-8859-1	pt	
ru	Russian	ISO-8859-5	ru	
sv	Swedish	ISO-8859-1	sv	
zh	Chinese	UTF-8	zh	

New row

Please note that removing a language will affect multi-language files written in that language. Adding the language again will restore normal access.

Enter the language codes in the order which they should be prioritized for multi-language files which lack content for a specific language. This list will be modified for each page request by taking the client browser's Accept-Language headers and language cookies into account.

en,de,fr,ar,da,es,fi,it,ja,nl,no,pl,pt,ru,sv,zh

OK Cancel

## Languages wizard

### Code

The ISO language code for this language. This is also the information you get when from the `<emit source="dir">` tag.

### Name

The name of the language. This will be used when the user chooses language.

### Encoding

The character encoding used for documents in these languages. Will be sent to the browser.

# 4 Links to External Files

---

The concept of Link to External File is similar to a symbolic link in Unix or a Shortcut in Microsoft Windows in that it is only a small file that points to the real file data.

## 4.1 Shared directory

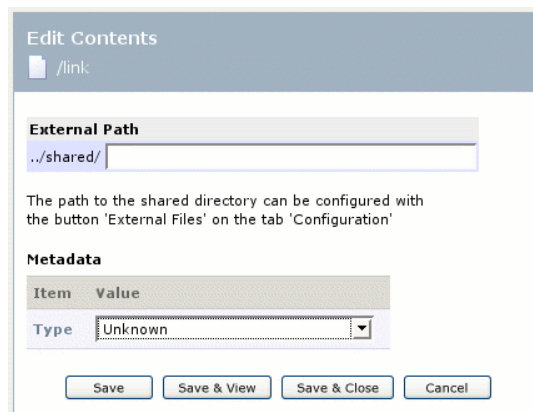
For security reasons all destination files are referenced with a path relative to a shared directory which defaults to <Roxen install dir>/shared.

The location of the shared directory can be configured in the SiteBuilder interface on the Configuration tab. Go to the Server Preferences tab and select the External Files...

## 4.2 Create a link

To create a new link, do the following:

1. Navigate to the directory where you want the new link.
2. Press File->Create New File.
3. Enter the name for the link and press Next.
4. Set the metadata Type value to Link to External File to create a link and press OK.
5. Select Edit->Edit contents... to bring up this dialog



6. Edit Link Content
7. Enter the name of the external file.
8. Select a type value corresponding to the type of the external file.
9. Press Save & close
10. Select Versions->Commit to commit the link file.

### Note!

It is only the small link file that is saved and versioned in the sitebuilder filesystem. The external file stays in the shared directory and is accessed when a request is made for the link file.

# 5 SmartSearch

---

## 5.1 Background

SmartSearch is a term used throughout Roxen CMS to describe a directory search strategy employed by the XSLT engine as well as other modules. The point of SmartSearch is to give you greater power and flexibility in building large sites. This is achieved by giving you a choice of where to place your files (templates or other types) within the site directory structure; depending on the location the site's behavior will vary accordingly, and you thereby get fine-grained control over the final result.

The fundamental idea is that Roxen CMS will look for resources, such as templates, in the same directory as the current page is requested from. If the template is found the search stops and that file is chosen. If not, the search continues in the parent directory and so on until the root level has been reached. It is normally considered an error if the requested file cannot be found after the root directory has been searched.

With this strategy in place you can design your site to take advantage of stationeries in a number of ways. Here are two examples:

### **Create modular layouts**

Place global XSL templates in the root directories and local variations, e.g. for a particular section or subsite, in the directory where they should be used.

Through `<xsl:import>` statements you can in many cases completely avoid code duplication. From time to time it may be necessary to add "hooks" to your template code, for instance calls to template rules which normally are empty. Next, in certain sections of your site you can then override these empty rules and perform whatever action you want.

Roxen CMS will take advantage of SmartSearch even if a template imports other templates in many levels, so with some planning ahead you can easily solve problems which otherwise would be far more difficult.

### **Stationeries**

A stationery is used whenever a new file of a given type is created. For many file types no stationery exist at all, and in some cases you may have more than one to choose from.

In Roxen CMS stationeries are searched for using SmartSearch. For example, this lets you place a press release stationery in the Press Release section and have all new pages in that subdirectory inherit content and/or metadata from this stationery without affecting pages created elsewhere.

When developing RXML code you can also call SmartSearch directly through the use of `<emit sb-search>`.

## 5.2 Performance optimizations

Although SmartSearch offers a large number of advantages there are also some costs to be aware of when building really large sites. For instance, in order to maintain cache correctness the system tracks all dependencies between content pages and other files they make use of (e.g. XSL templates). This means that many directories may potentially contain files which will affect the result of a SmartSearch

operation, and therefore need to be considered a dependency in case a template is added there in the future.

These "virtual" file dependencies (as well as dependencies to actual files) will in turn increase the likelihood of cache regeneration for the original page, since any file operation in those locations can potentially affect the page in question. This is where knowledge supplied by the site developer can help Roxen CMS cache system in a very important way: by telling the system that you will never use templates from a particular directory you let the system ignore any dependencies to that directory.

The details on how to configure the XSLT/SmartSearch interaction is found in the online help of the Administration interface in the Sites -> (your site) -> CMS: XSLTransform -> Settings tab.



# 6 Application Launcher

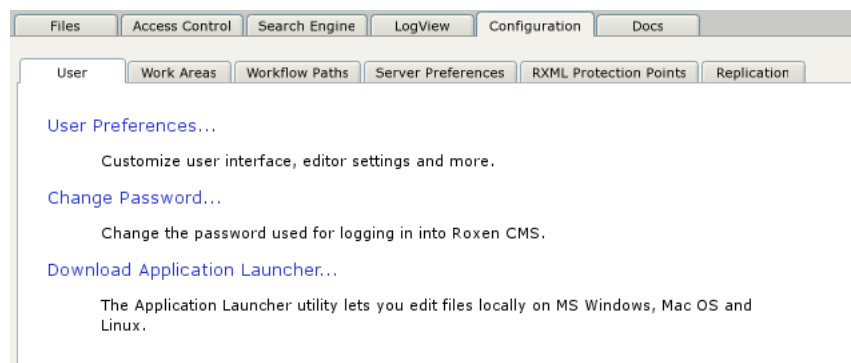
---

Roxen Application Launcher is a program that lets the user use any program to edit files handled by SiteBuilder. There are three different incarnations of the Application launcher; the Mac version, Windows version and the Unix version.

## 6.1 Windows

The Application Launcher may only be installed by a person with Windows administrator privileges. This is to ensure that it will be setup according to company policies. Also, the administrator might want to make sure that only approved programs may be launched.

The Windows version of Application Launcher uses the HTTP/HTTPS protocol for communication. This implies security hazards if the server is not setup properly. The user might not notice any of this but the consequences may be dire. One thing the user will notice though is if the site does not have a valid Thawte/VeriSign certificate when running HTTPS. A self signed or otherwise invalid certificate will make the browser generate a security alert message which might make the user uncomfortable as she does not know if his connection is secure or not.



Click the "Download Application Launcher" link.

The Application Launcher install program can be downloaded from the CMS Advanced server by clicking on the Download AppLauncher button under the Configuration tab in the Content Editor. Execute the file application\_launcher.exe and choose an appropriate directory for installation. The default directory supplied by the install program is usually a good choice.

If the Application Launcher has been properly installed, it can be started from the Programs menu (without editing a file in the Content Editor). The interface is available from the system tray on the Windows taskbar when up and running.

All changes made to a document are saved in a unique temporary directory. Each document has its own directory. The Application Launcher polls this directory through Windows and sends the document changes to the Content Editor as long as the Application Launcher is running. The changes are only sent when a Save operation is done inside the application.

Application Launcher stores all files in a temporary directory which differs slightly depending on what operating system and language is used (see below).

## Windows 2000

Usually the Document settings directory:

```
C:\Documents Settings\"user_id"\Local Settings\Temp\AppLaunchTemp-Dir\"random generated unique directory\"filename.extension.
```

## Windows NT 4

Usually the Windows installation directory:

```
C:\WinNT\Temp\AppL...\filename.extension.
```

For an overview of the Application Launcher interface and a troubleshooting guide, see the Content Editor Manual.

## 6.2 UNIX

First off, the requirements; to use the Unix version of the Application Launcher, you need a file system shared between the server and the client machines from which your users access the server. This is typically achieved through external file sharing programs, such as Samba or NFS. On your client machines, you also need a pike installation, the Unix Application Launcher found in server-[version]/tools/roxen\_launcher.pike, and its configuration file. Lastly, the browser must be set to invoke the Application Launcher for its associated content-type.

When the user chooses to edit a file, SiteBuilder copies it to a shared directory and sends a response to the user's web browser. The response has the content-type application/x-roxen-launcher and contains the path to the file and its actual content-type. If Application Launcher has been installed and the web browser properly setup, the browser will launch the Application Launcher, which in turn starts the editor appropriate for the file being edited. The application and its startup command line (including the path to the file) is executed as configured in the launcher configuration file.

Several actions are necessary for this to work:

## 6.3 Installing the Application Launcher

The Application Launcher must be installed on each user's computer. Application Launcher exists in a Windows and a Unix version. Both versions can be found in the tools/ directory of the Platform distribution, and the Windows version is also downloadable from the Configuration tab of the Content Editor.

The Windows version is distributed as a self-extracting .exe file. It should install Application Launcher as well as do the necessary configuration for it to be recognized by any web browser. It is however often necessary to configure it further, see the Local editors section for how.

### Note!

The application launcher can be sensitive to old Windows programs that are not written for a 32-bit architecture. If you have problems installing the application launcher, try exiting any older applications you may have running, and this will usually solve the problem.

The Unix version is a simple pike script, tools/roxen\_launcher.pike. It needs to be installed on the Unix system and each web browser must also be installed to recognize it. In Netscape this is done in the Navigator/Applications part of

Edit/Preferences. The MIMEType should be application/x-roxen-launcher, the application /usr/local/bin/roxen\_launcher.pike %s.

The Unix version is configured through .roxen\_launcherrc files. By default the user's home directory as well as /etc/ will be searched for a .roxen\_launcherrc file. See the .roxen\_launcherrc section for the syntax.

## 6.4 Local Editors

The Local editors wizard is used to create registry configuration files for the Windows version of the Application Launcher. The user needs to download such a file and double click on it to install that configuration.

The configuration contains a content-type followed by either a file extension or the full path to a program. Wild cards can be used to some extent, the content-type image/\* and \*/\* are valid but not image/g\* nor \*/gif.

Application Launcher will match the content-type it got from SiteBuilder against the content-types in the configuration, one line at a time. When it finds a match it will try to start the appropriate application. If a file extension is given it will try to start the application that handles that file extension. If a path is given it will try to start that program.

The Download editor button will download the .reg file for the current configuration. Double-clicking on the .reg file will read the configuration into the registry.

The reason that the installation needs to be done in the registry, rather than being part of the editor profile, is to gain security. The web server should not be able to start any program on the user's computer, only the programs that have been configured as editors.

## 6.5 .roxen\_launcherrc

The .roxen\_launcherrc file is used to configure the Unix version of the Application Launcher. As with most Unix applications each user can have their own configuration. A sample .roxen\_launcherrc file can look like this:

```
text/* emacs $nfs
image/* gimp $nfs
```

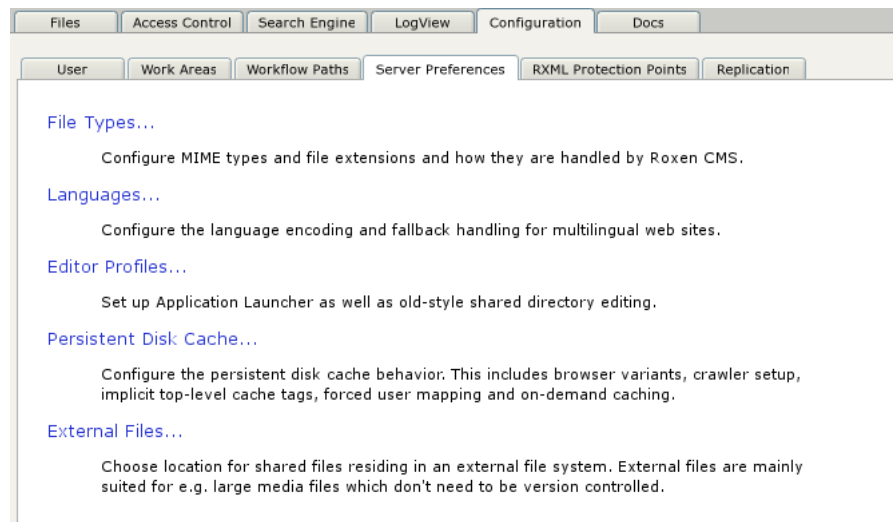
Each line contains a content-type followed by the program that should be used to handle that content-type. The content-type uses the same limited type of wild cards as the Windows version. \$nfs will be replaced with the path to the file that are to be edited.

### Note!

The word emacs in the example tells Application Launcher to launch one instance of emacs for each file. If emacs in the example would be substituted with emacsclient (or gnuclient for xemacs) then each file being edited would have a buffer in one instance of emacs. This does however imply that the user already has an instance of emacs up and running, otherwise nothing will happen.

## 6.6 Editor Profiles

The bulk of the configuration is done in the Editor profiles wizard located . Here several different editor profiles can be configured, each user can then choose one of those profiles. It will however usually be sufficient with one editor profile.



### *Editor profiles link*

The access for local editors comes in two flavors, anonymous and named. Anonymous is used for users that exist in SiteBuilder, but not on the operating systems user database. Named is used for users that are imported into SiteBuilder from the operating systems user database.

Anonymous access works by writing the shared files to a directory that is not readable to anyone. SiteBuilder will create a subdirectory with each file in. That sub directory will be readable to anyone and the file both readable and writable to anyone. But the name of the sub directory will be known only to the user. It is very important that it is not possible to find out the name of the sub directories by listing the shared directory.

The anonymous access is configured through the following variables:

#### **Enabled**

Whether the anonymous access should be enabled at all.

#### **Directory**

The directory where the server will create its sub directories. The directory must be created by the administrator and it should not be possible for anyone to read it, not even the server itself. Of course, the user running the server must have write permission here, so it can create its files and sub directories.

#### **SMB path**

The path from which the client can access the directory using Windows file sharing. If running on Unix, a file sharing program such as Samba will have to be used to make it available. The sharing should be configured to be available without password.

#### **NFS path**

The path from which the client can access the directory using NFS. Only necessary if the Unix version of Application Launcher is used.

#### **AppleShare path**

The path from which the client can access the directory using AppleShare. Some sort of AppleShare sharing software will have to be used. Only necessary if the Macintosh version of Application Launcher is used.

Named access works by writing the actual files as the user who owns them. This will only work for users that have been imported from the operating system to SiteBuilder. Furthermore, the server has to be run as root, otherwise it will not be able to create files as another user.

SiteBuilder will create a sub directory for each user who uses the named access. These sub directories will be created with the correct file permissions so that only the appropriate user has access to it. Within each sub directory, a directory structure matching the site will be created.

The named access is configured through the following variables:

#### **Enabled**

Whether the named access should be enabled at all.

#### **Directory**

The directory where SiteBuilder will create its files. Unlike the anonymous access, this directory may well be readable by anyone. File accesses will be protected by the operating systems access control system.

#### **SMB path**

The path from which the client can access the directory using Windows file sharing. If running on Unix, a file sharing program such as Samba will have to be used to make it available. The sharing should be configured so that each user has to log on with user name and password.

#### **NFS path**

The path from which the client can access the directory using NFS. Only necessary if the Unix version of Application Launcher is used.

#### **AppleShare path**

The path from which the client can access the directory using AppleShare. Some sort of AppleShare sharing software will have to be used. Only necessary if the Macintosh version of Application Launcher is used.

When the anonymous and/or named directories have been configured, it is time to configure one or more editor profiles. Usually one editor profile will be suitable. Each editor profile contains one or more editor definitions. The typical reason for more than one editor profile and more than one editor definition is to enable SiteBuilder to be user together with thin client solutions like WinFrame or Tarantella, or in some very heterogeneous environment where clients must use different solutions.

Each editor profile contains a name and whether it is valid for anonymous, named or both anonymous and named access. They contain the following variables:

#### **Valid for content-types**

This is a comma separated list of content-types that can be edited by this editor definition. The now familiar limited type of wild cards can be used. Each site will probably need to configure this to reflect the file types used on the site.

#### **Output file content-type**

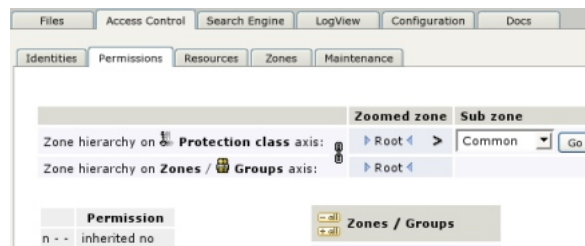
The content-type of the file sent to the browser when the user chooses to edit a file. Should be set to application/x-roxen-launcher if the Unix Application Launcher is used.

#### **Output file contents**

The contents of the file sent to the browser when the user chooses to edit a file. If the Unix Application Launcher is used the file contents should be:

```
nfs=#nfs#  
smb=#smb#  
appleshare=#appleshare#  
content-type=#content-type#
```

# 7 Access Control



## Example from the Access Control tab

Access Control is the permission handling system in SiteBuilder, allowing detailed configuration of different users' access to different parts of any SiteBuilder site as well as of the SiteBuilder system itself.

Access Control handles different users' permissions to certain protection points. A protection point is a resource being protected, for example a directory or the Content Editor. A user has read, write or no permission on each protection point.

The permissions are indirect. Users have membership in groups, the groups then have permission on protection classes. Each protection class contains one or more protection points. Thus users belong to groups while protection points belong to protection classes.

The Access Control interface is reached via the Access Control tab in the content editor. When selecting the tab, five additional tabs will be visible; Identities, Permissions, Resources, Zones and Maintenance.

### Identities

The Identities tab is where Users and Groups are handled and group membership assigned.

### Permissions

The Permissions tab is where permissions are set for groups on protection classes.

### Resources

The Resources tab allows for grouping of Protection Points into Protection Classes, as well as configuration of the protection points and protection classes.

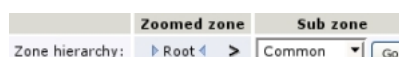
### Zones

The Zones tab controls the administrative grouping of users, groups and protection classes into Zones, for simpler administration.

### Maintenance

This page contains the possibility to purge references to removed modules from the database and create a file dump of the AC database.

## 7.1 Zone Navigation



At the top of most pages inside the access control system is a zone navigation widget. It can be used to limit the amount of visible information in the zone tree below it. By default it is set to the top of the zone tree (the root). Selecting a sub zone will cause the tree below to use that zone as its current root, hiding the parent zone and any siblings. Clicking on a parent zone in the zone navigation will bring back that zone as the root zone.



The Identities and Permission pages has zone trees on both axis and hence a double zone navigation. Usually they are tied together and navigated as one. By clicking the chain symbol, the zone trees will be unlocked and can then be controlled individually, making it possible to show different zone trees on the two axis. Clicking the broken chain symbol will tie them together again.

## 7.2 Users and Groups

Users, groups and memberships are handled on the Identities tab. It consists of a table with users and groups to the left and the groups they can be members of on the top.



- M explicit membership
- m indirect membership
- a automatic membership
- ( ) explicit non-membership

View ▾ Users and Groups

Zones / Groups

Zone	Groups	Root	Common	Content	Employees	Partners	System
Root (5)	0 groups						
Common (0)	1 group						
Everyone ( )		x	x	x	x	x	x
Content (0)	2 groups						
Content Administrators ( )		M	x	-	-	-	x
Content Authors ( )		M	-	x	-	-	-
Employees (0)	1 group						
Employees ( )		M	-	-	x	-	-
Partners (0)	1 group						
Partners ( )		M	-	-	-	x	-
System (0)	3 groups						
Administrators ( )		Mm	M	-	-	-	x
Employee Administrators ( )		M	-	-	-	-	x
Partner Administrators ( )		M	-	-	-	-	x

Zones / Users

Zone	Users	Root	Common	Content	Employees	Partners	System
Root (5)	0 users						
Common (0)	0 users						
Content (0)	0 users						
Employees (0)	0 users						
Partners (0)	0 users						
System (0)	1 user: Zoom: Adm- ▾ Go						
Administrator (stars)		Mm	m	-	M	-	M

Search:

Search  Users  Groups

Memberships are assigned by clicking on the squares in the intersection between the users or groups to the left and the groups they become members of on the top. A click will change the membership state, which is indicated by a letter. It might take several clicks to circle to the wanted state.

The different states are:

- Not a member.

M Explicit membership, set by the user.

m Indirect membership, given because this user or group is member of another group that has an explicit membership.

M Both an explicit and indirect membership.

m

a Automatic membership, imported from another source.

( Explicitly not a member. Used to counteract an unwanted automatic

---

) membership.

---

x Not possible to make this group a member, since two groups cannot be members of each other. The group Administrators cannot be a member of the group local because local is already a member of Administrators.

---

### 7.2.1 Users

A user in SiteBuilder contains a full name as well as a user name used when to log on. The password is handled by an access control authentication module. Each user can have more than one authentication module or even more than one copy of an authentication module. That way a user could have an additional password by having an extra copy of the Internal Password module.

Users are often imported from another source, such as the operating system's user database. If so, the user configurations are done in the module importing the users.

It is however also possible to create users directly in the Access Control interface. It is done by pressing the New user button, and filling in the user name as well as the user's full name. By default a copy of the Internal password authentication module will be added as well. It is used to give the user a password to log in with. It is possible to write the password itself, or give a version encrypted with the Unix crypt function. The later is good for sharing passwords between different sites.

Users that are created directly can be mixed with imported users. That way users from within the company could be imported while users from customers are created in the access control interface.

To later change or remove an user, just click on the user name in the user listing on the left.

### 7.2.2 Groups

Groups contain members that are either users or other groups. As with users groups can be imported from other sources, such as the operating systems user database, or created directly in the access control interface. Memberships are imported together with the groups, thus making it possible to import the entire security settings from another source. It is still possible to change memberships to imported groups in the Access Control interface. Such changes will however be local to the Access Control interface, they will not be exported back to the original source.

There is a special group, Everyone. Requests that could not be linked to any user, such as a request with no authentication information or a request with the wrong password, will get the permission of the Everyone group.

To create a new group press the New group button and give a name to the group.

## 7.3 Protection Points

A protection point is a resource that is handled by the Access Control system. The resource can be a directory, a single file, the graphical interface, the Access Control system itself or a work area.

SiteBuilder automatically creates a number of protection points when the system is set up or a new work area is created, and also when some additional modules are installed on the server.

It is also possible to create protection points from the Content Editor both for files and directories and for use in RXML. A protection point for a file is created or

removed by choosing Edit permissions... in the Access menu under the Files tab. A protection point for use in RXML is created, modified or deleted by clicking on the Create RXML Protection Points buttons (List, Create, Change, and Remove) on the Configuration tab page.

### 7.3.1 Protection Points in External Modules

Other modules, not included in the SiteBuilder distribution, might also add their own protection points to the Access Control system. The modules' protection points will show up automatically in the Access Control configuration interface when the module is added to a SiteBuilder server.

## 7.4 Adding Protection Points

In addition to the protection points created automatically by SiteBuilder it is possible to create additional protection points.

### 7.4.1 Protection Points in File Structure

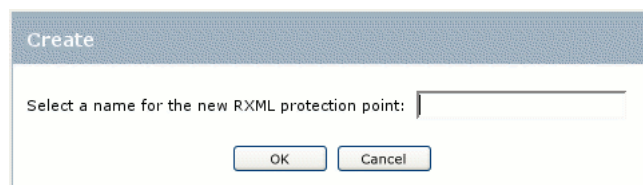
Protection points can be created for directories or single files in the virtual file system of SiteBuilder.

Use the Edit Permissions... command in the Edit menu to define access permissions for the given file or directory. More information on this wizard is found in the section Edit Permissions.

### 7.4.2 Protection Points in RXML

It is possible to create protection points to be used inside the content and template files. They are handled through the Configuration tab in the Content Editor.

There is a special protection point controlling permission to create RXML protection points named RXML protection point administration.



#### *Creating a protection point*

Such a protection point can be used together with a `<if ppoint=>` tag placed around restricted information included in a web page.

## 7.5 Permissions

The No rights, Read and Write permissions can be set between groups and protection classes. All members of a group will inherit the permissions of the group for each protection class.

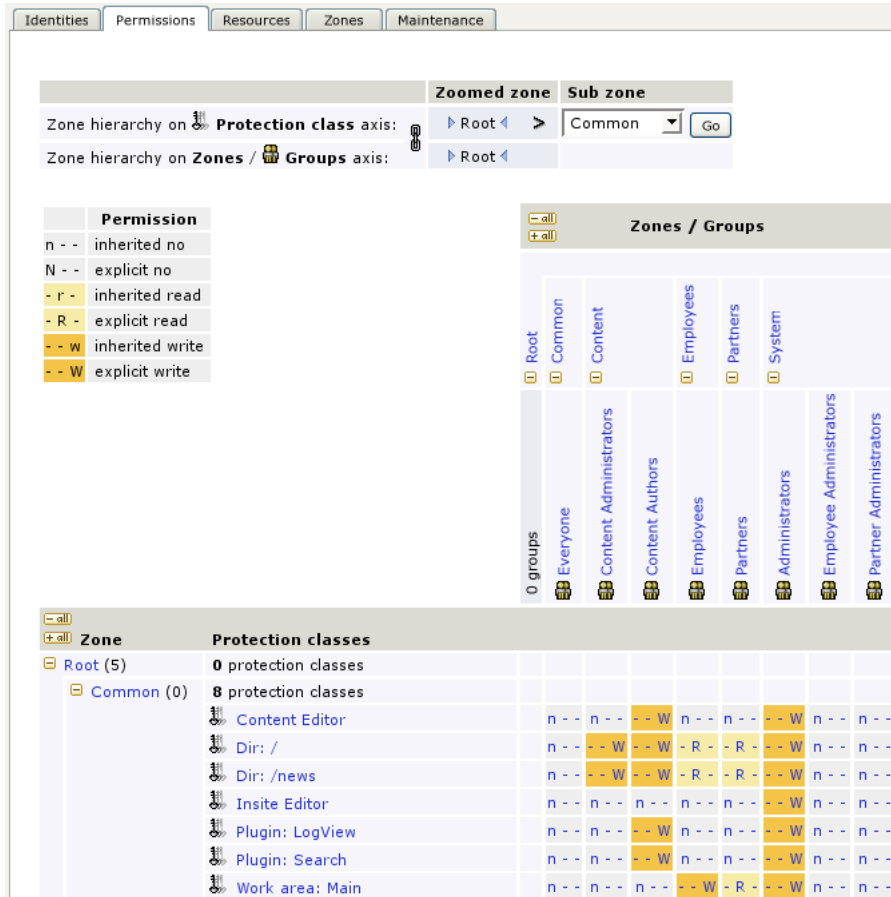
- No rights mean that names of files and directories can not be viewed or changed by the user.
- Read permissions mean that the user is able to view but not to change the name of the files and directories in the directory.

- Write permissions mean that the user is allowed to view and change the names of the files and directories in the directory.

**Note!**

It is assumed that Add and Remove are special cases of "change".

If a user is a member of several groups and these groups have different permissions for a protection class, the user will receive the most privileged permission.



**7.5.1 Setting Permissions**

Permissions are set by clicking in one of the three columns. Clicking in a column will toggle permissions between a hyphen (permission not set), lower case letter (inherited permission) and capital letter (direct permission).

There are three kinds of permissions to be set between a group and a protection class (see table below).

Marking	Permission type
n - -	No rights, inherited permission
N - -	No rights, direct permission
- r -	Read, inherited permission
- R -	Read, direct permission
- - w	Write, inherited permission

-- W	Write, direct permission
------	--------------------------

## 7.6 Protection Classes

A protection class is essentially a group of protection points. They are used to simplify handling of several protection points that should have the same permissions. A protection class will however often contain only one protection point.

### 7.6.1 Resources

On the Resources page the protection classes can be configured. Protection points can be moved between protection classes.

Zone	Description	From	To	Delete	Owning module
Root (5)	0 protection classes				
Common (0)	Content Editor (1)				
	Content Editor				CMS: Main Module
	Dir: / (1)				
	Insite Editor (1)				
	Plugin: LogView (1)				
	Plugin: Search (1)				
	Work area: Main (1)				
Content (0)	1 protection class				
Employees (0)	0 protection classes				
Partners (0)	0 protection classes				
System (0)	9 protection classes				
	Move to a new protection class.				
	<input type="button" value="Create New Protection Class"/>				<input type="button" value="Move"/>

#### Section of the Resources page

SiteBuilder automatically adds a number of protection points with separate protection classes which are used to control access to the different parts of SiteBuilder.

A typical editor would need write permission for Content Editor, at least for one work area and one or more Dir or File protection classes to be able to create or edit files.

For a public site, the group Everyone needs read permission to a work area and one or more Dir or File protection classes.

## 7.7 Predefined Protection Classes

The following protection classes are predefined in SiteBuilder:

### 7.7.1 Module specific protection classes

#### Booking Administration

This protection class gives a user full administration permissions to the Bookings module. An administrator can create a resource and edit current resources. An administrator can also reschedule and delete bookings. This protection class is dependant on the Bookings module.

### **Booking '<resource name>' admin**

This protection class gives a user administration permissions for a specific resource. A resource administrator can delete or reschedule any booking and edit information for the given resource. This protection class is dependant on the Bookings module.

### **Booking '<resource name>' book**

This protection class gives a user permissions to make a booking for the specific resource, delete or reschedule owned bookings. With this protection class a user can also review other users bookings for the given resource. This protection class is dependant on the Bookings module.

### **Forum (<id>) Administration**

This protection class gives full administration permissions to all created forums. Forum administrators can, besides basic and admin forum privileges create new forums and edit current forums. This protection class is dependant on the Forum module.

### **Forum (<id>): admin**

The forum admin class main function is handling threads and messages. A forum admin can move, copy, delete, hide, and lock threads. They can also delete and edit other users messages. Admins can also do IP-bans. Setting read permissions at this protection class removes top-level admin privileges, except IP-bans. This protection class is dependant on the Forum module.

### **Forum (<id>): compose**

Write permissions in this class gives a user access to all basic forum capabilities. These are: starting a new thread, posting messages, editing previously posted messages and reply to other messages (with the option of quoting previous message). Setting read permissions will allow the user to browse existing threads and read messages. This protection class is dependant on the Forum module.

## **7.7.2**

### **Default protection classes**

#### **AC zone: Root**

A new protection class will emerge for each zone that is created.

The protection class controls permission for the Access Control interface. Read permission is required to be able to see the Access Control information. Write permission is needed to make any changes to the Access Control information.

The special class AC zone: Root always exist and controls all underlying zones. Compare Dir: /.

#### **Content Editor**

Controls access to the Content Editor, the graphical interface of SiteBuilder. To change anything, edit files, changing permissions or settings, in the SiteBuilder, write permission for this protection class is required.

#### **Dir: /**

This protection class controls the access to the files and directories of the site. It is also possible to create new protection points for single files or directories. A protection point for a directory protects all files and subdirectories of that directory, unless a file or subdirectory has a protection point of its own. Thus the protection point for Dir: / will protect all files and directories until protection points for other directories and/or files are created.

**Note!**

Dir: / is the only directory protection point that is not possible to remove.

New protection points for files and directories are created with the Edit permissions wizard under the Files tab.

On a public site the group Everyone should have read permission for Dir: /.

**Insite Editor**

This protection class controls access to the Insite Editor toolbar. If set to read permissions the toolbar will show, but all buttons will be grayed out. Setting non-permissions disables the Insite Editor toolbar completely.

**Plugin: LogView**

Users with read permission to this protection point can view statistics under the LogView tab. Users with write permission can configure LogView. This protection point will be generated automatically when the LogView module is installed and will then copy the permissions from the Content Editor protection point.

**Plugin: Search**

Users with read permission to this protection point can view the settings under the Search Engine tab in SiteBuilder. Users with write permission can alter the different settings in the Search Engine. This protection point will be generated automatically when the Search: SiteBuilder Interface module is installed and then copy the permissions from the Content Editor protection point.

**Purge files/directories**

Purge removes files or directories permanently from SiteBuilder's file system with no way of restoring the data using the tools in the Content Editor. To be able to purge files or directories from SiteBuilder's file system, write permission is required for this protection class.

**Work area administration**

To be able to create a new work area or change settings for the current one, write permission is required for this protection class. It also covers administration of SiteBuilder preferences and replication.

**Work area: Main**

There will appear one new protection class for each work area that is created.

The Main work area is created automatically when setting up SiteBuilder. It controls access to the files in the Main work area. Whenever a new work area is created, another protection class is also created for it.

The protection point for a work area is combined with the protection point for a file, to find out what permission a user has to a file in a certain work area. The combined permission equals the lowest permission of the two combined protection points. Thus a user will have read permission to a file in a work area she has read permission to, even though she has write permission to the protection point protecting the actual file.

Editors need write permissions for at least one work area to be able to edit files.

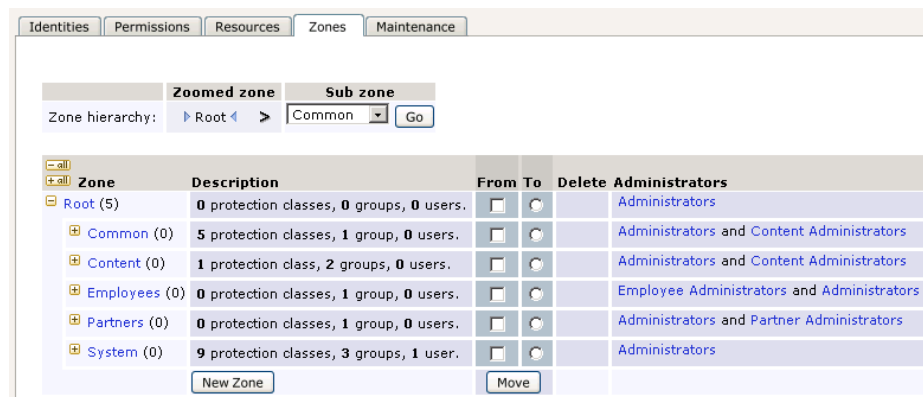
To make a site accessible in public the group Everyone needs read permissions to the work area containing the live site. Which files will be accessible is controlled by Dir and File protection classes.

## RXML Protection point administration

To be able to create new RXML protection points, remove or edit existing ones, write permission is required for this protection class.

## 7.8 Zones

To make large sites (with many identities or protection classes) manageable, identities and protection classes in the Access Control system are grouped into zones, with each identity and protection class belonging to exactly one zone. On the pages in the Access Control area the zones can be folded or unfolded, or only parts of the zone tree can be viewed.



### Zones tab page

By default, the zone "Common" is created containing all identities and protection classes. Additional zones can be created by clicking on the Add zone button, entering a name for the zone, setting parent zone and then clicking on Add to create the zone. Identities or protection classes can then be added to the zone by moving them from another zone.

Moving is done by checking the groups, protection classes or zones you want to move in the From column. Selecting the zone you want to move it to in the To column and clicking the move button.

For each zone a protection point will be created. A user will only be able to see the zones he has read or write permissions for. To be able to move identities between two zones, write permission is required for both zones.

Zones should not be confused with groups; zones primarily affect the administration of the Access Control system, and have no direct impact on permissions checking. Zones do affect it indirectly, though, since they allow delegation of some Access Control administration to other users than the main administrator.



# 8 Persistent Cache in SiteBuilder

---

The persistent cache is a performance optimization for sites which have large number of read-only page accesses.

The traditional way of serving a page in SiteBuilder is to first apply the XSLT template and then process any RXML commands in the page. The first step, XSLT transformation, is normally time-consuming and have relied on RAM caching for speed. In other words, the result of the transformation has been kept in memory in case another visitor requests the same page in the near future.

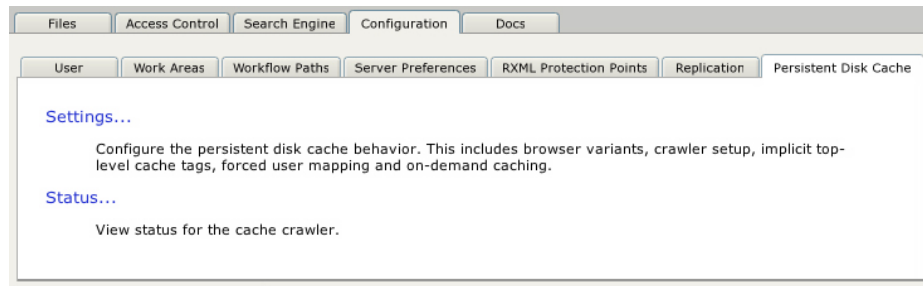
After XSLT template application the RXML commands need to be processed. Since RXML normally is highly dynamic the output of this processing cannot be cached efficiently. This meant that the RXML processing has to start over for each and every visitor, even if the same page is repeatedly requested.

RXML commands are compiled into an internal representation which is more efficient for repeated execution. Moreover, the output from the XSLT transformation (in the form of compiled RXML) is now stored persistently on disk. This means the work of applying a template and compiling the RXML commands is now done once (on the backend server), and the resulting cached version is distributed to all frontend computers in a multiserver configuration. The benefit from this is that the frontend machines are able to serve pre-cached files even if they are restarted, thus avoiding the performance hit caused by empty RAM caches.

For the caching system to work efficiently a number of restrictions must be followed:

- The pages must be careful about dynamic behavior in the XSLT template. Examining e.g. browser type in the template is still possible, but this requires that the prefetch crawler which generates the disk cache is configured to ask for the page in all expected variants. Likewise, having a template rely on e.g. form variables means that all possible input values cannot be known, thereby reverting to on-demand transformation and lower performance.
- The XSLT implementation has been extended with a `<rxml:variable-dependency/>` tag which lets the stylesheet author control precisely which form variables the stylesheet depends on. Using this lets the persistent cache system share cache entries more often even though RXML code on the same page have other variable dependencies.
- The prefetch crawler will only access view area pages as user Everyone. This means the pages which use XSLT to customize pages based on user identity cannot benefit from this caching.
- RXML compilation is also rather aggressive about caching by default. Code which must be re-evaluated for different users, browsers, domains, cookies, form variables etc. needs to be identified using `<cache>...</cache>` tags with appropriate attributes.

## 8.1 Setting Up



The Configuration tab in SiteBuilder has a tab dedicated to actions related to the disk cache. Click the "Persistent Disk Cache" tab to access the following two wizards:

### Settings

Control settings related to the persistent disk cache.

### Status

View pages queued for processing in the disk cache.

## 8.2 Enable/disable persistent disk cache

This switch can be used to enable or disable persistent caching. Enabling the cache will not in itself trigger any crawling of all files on the site. Instead a page requested from the view area will be transformed and compiled on-demand at the time of the next request, but will then simultaneously be saved into the persistent cache for retrieval in all following accesses.

However, once a page, or data it depends on, is changed in the Content Editor, the crawler will be used to update the cache. For example, committing new content for a page which is currently in the disk cache will notify the prefetch crawler which will access the page to make sure the newest content is placed in the cache.

## 8.3 Crawler configuration

Since the prefetch crawler needs to emulate visitors on the site in order to generate cached pages, the administrator should configure the crawler to behave as those web browsers the site design is adapted to.

It is important that only those browsers for which it is necessary to present adapted content is listed here. If the site contains pages which, say, send different JavaScript code based on browser type, that distinction must also be reflected in the crawler so that both types of JavaScript are cached instead of only one. On the other hand, if the pages don't require that distinction it would only be a waste of processing power and disk space to generate and cache multiple but identical copies of the pages.

Since the number of actual browser types in the world is vast, the wizard offers a way to map many browsers into the ones you list. By doing this you can ensure that even if pages depend on browser type, the browser identification string your RXML or XSLT code actually see is one of the ones in this list. In other words, the administrator can ensure that the site will only serve cached files.

The table in the configuration wizard has four columns. The fourth one, "Map to browser", holds the browser identification strings that are used to locate the

appropriate cache entry when there is a page request. The two preceding columns are used for entering regular expressions which translate the true browser identification string to the internal ones. The entry titled "fallback entry" cannot be deleted and represents the settings used if no other entry matches a request.

**Note!**

It is possible to disable caching around certain parts of the RXML code with the `<nocache>` tag. The browser type that the code will see in such parts is the real one in the request. The browser type is only remapped for purposes of the "browser" cache profile in the RXML `<cache>` tag (see "RXML compilation"); it's not remapped in the actual requests.

The first column, "Crawl", controls which of the browser types that should be prefetched. Enabling every entry means that all possible browser variants will be prefetched. Consequently, disabling all will in effect disable the crawler. Note that there is a restriction in that cache files generated by the crawler will not be expandable by on-demand caching later, so enabling more than one but less than all crawlers is normally not a good strategy.

## 8.4 Prefetch restrictions

Committing a change to one XSLT template used on a large number of pages will naturally invalidate all those pages from the cache. Since regeneration of cache contents can add significant workload to the backend server the administrator can restrict the prefetch in various ways.

The "Path include regexp" should hold a regular expression which identifies the files that the crawler should process. If left blank all paths are used by default. Similarly, a path exclusion regular expression may be provided in the field "Path exclude regexp". Note that only files with the content types XML and HTML are of interest to the crawler so all other types are automatically filtered.

The "Maximum crawl time" parameter can be used to limit the number of minutes that the crawler runs whenever a change affecting a large number of files has been made. Setting e.g. a ten minute limit would let the crawler complete as much work as possible for a duration of ten minutes, possibly leaving the remaining files for on-demand transformation. The crawler will prioritize files higher up in the site hierarchy which means that files in the root directory will always be completed first.

## 8.5 Top-level caching

SiteBuilder normally wraps a `<cache>` tag around every XML and HTML page to gain additional performance. This option lets you disable this tag or alter the "profile" attribute used as the tag argument. The default "browser" selection separates the cached data based on the "Map to browser" identification strings. See below for more information on the consequences of using `<cache>` tags.

## 8.6 Force access as Everyone

Sites which adapt their contents to different user identities would get little benefit from persistent caching since there would be as many variants for each page as there are users. It's avoided by using this option to let all authenticated users to share the same cached pages. Any contents which needs to be user-specific must be wrapped in `<cache>` tags to disable caching for those particular sections.

The following options are available:

#### **Force access as Everyone**

The crawler runs using the Everyone identity. For this to work you need to open up the work area and its directory protection points for Everyone.

This setting is primarily suited for web sites where external users can access pages without authorization.

#### **Force access as everyone, but crawl as fully privileged user**

The crawler will run using an Access Control backdoor which gives it full access to every page in the site, thus generating cache data even for pages which require authorization. However, when a site visitor asks for a page, each request will be filtered based on that user's permissions.

This setting is suitable for e.g. intranets where user Everyone normally is denied access to the web site.

#### **Note!**

The request filtering is on a page-by-page basis meaning that if a page is considered to be ok for viewing, all content on that page will be returned from the cache. This means that subsections within the page where more restrictive permissions are needed must be wrapped in <nocache> tags to enforce the restrictions in a dynamic way.

#### **Access as each user**

Similar to the first option, this one also runs the page crawler using the Everyone identity, but incoming page requests for authenticated users will not get cache hits for entries generated as Everyone. Instead the cache will keep separate entries for each combination of user permissions, resulting in on-demand cache updating and higher storage requirements.

Use this setting if the site has restricted contents which hasn't been wrapped in <nocache> tags.

## 8.7 First time setup

The prefetch crawler will run when it detects a cached file which has become stale. It is also run when a new file is committed to the repository. Files in need of prefetching will be placed in a queue which the cache system processes periodically.

However, this leaves the administrator with a "bootstrap" problem where the cache won't be completely filled when an existing site is migrated or if changes are made which do not automatically notify the prefetch crawler (e.g. renaming a user in AC). To handle this are two actions in the Admin menu in the Files view of the Content Editor: "Disk cache: Prefetch tree" and "Disk cache: Prefetch file". The former will add a directory and all of its content to the crawler queue, and the latter will add a specific file to the queue. These actions are only available to those with write access on the protection point "Work area administration".

## 8.8 RXML Compilation

The RXML parser in both WebServer and SiteBuilder 2.2 will now compile RXML commands before execution. Previously the parser had to scan for e.g. the character sequence "<", "d", "a", "t", "e", "/", ">" to detect a reference to the <date/> tag, even when this tag was placed inside a loop which was run more than one iteration. Now

the parser will reuse internal data, called p-code, more efficiently whenever the same RXML code is executed more than once.

Further speedups can be realized by wrapping code sections inside `<cache>...</cache>` tags. This will cache the output generated by the RXML tags. However, using `<cache>` means that the tags are only executed once and then a "snapshot" of their output is taken, so inserting that snapshot at another request for the page may produce unwanted results if the tags actually depend on other data in the environment (browser type being one example of this). The `<cache>` tag has been constructed to take attributes which solve these issues, but identifying and applying them is a manual process which is left to the page author, i.e. there is currently no system to automatically deduce the proper cache parameters depending on the tags and variables in the content.

If there are nested `<cache>` tags in the content of a `<cache>`, they are (normally) cached separately, and they are also recognized so that the surrounding `<cache>` tag does not cache their contents too. It is thus possible to change the cache parameters or, with the `<nocache>` tag, completely disable caching of a certain part of the content inside a `<cache>` tag. Note that this implies that any RXML tags that surrounds the inner `<cache>` or `<nocache>` tag will not be cached, since the outer `<cache>` can not cache their results without caching the results of the inner `<cache>/<nocache>` too.

Besides the value produced by the content, all assignments to RXML variables in any scope are cached. I.e. an RXML code block which produces a value in a variable may be cached, and the same value will be assigned again to that variable when the cached entry is used.

#### **Note!**

Note that it's easy to create huge amounts of cached values if the cache parameters are chosen badly. E.g. to depend on the contents of the form scope is typically only acceptable when combined with a fairly short cache time, since it's otherwise easy to fill up the memory and disk on the server simply by making many requests with random variables.

Since the caches are persistent it's very important to limit their size so that every page doesn't generate more than around, say, five or ten different cache entries. The `<cache>` tag has a `profile` attribute to aid in this. It's used to choose a predefined cache configuration, which typically contains provisions to not grow without bounds (as opposed to caching on a forms variable).

SiteBuilder provides two cache profiles:

#### **browser**

The cache key is one of the browser names in the "Map to browser" column in the persistent cache wizard. Thus there won't be more cache entries than there are browser alternatives.

This is the default top level cache profile around all RXML code, but it is configurable with the "Top-level caching" setting in the persistent cache wizard.

#### **userperm**

The cache key represents the access permissions of the current user. This is useful to cache things that depend on the files and directories different users have access to, typically navigation menus generated from lists of subdirectories. The number of alternatives is bounded by the number of different access combinations in AC - usually the same as the number of groups.

The p-code that is generated from RXML and stored in the persistent cache contains references to the actual functions and classes that are executed when the p-code is reused. It's therefore highly dependent on the currently loaded tag modules. Reloading a tag module will invalidate all the RAM cached p-code in the same site, but it won't affect the disk cache. However, adding or removing a tag module will cause all the p-code on disk to be invalid when it's loaded, which effectively causes the whole persistent cache to be invalid and therefore regenerated on demand.

This also has implications in a multiserver setup: If a tag module exists on a backend but not on a frontend, then all p-code that contain tags defined in that module cannot be used on the frontend. The p-code will be replicated as usual, but the frontend will consistently fail to decode it and therefore fall back to on-demand caching.

The consistency check is however a little more lax on a frontend: Adding or removing a tag module won't invalidate all the p-code in it. If a module is removed, it will only invalidate those entries that actually contain tags from that module. On the other hand, if a module is added and there are tags in the RXML pages that it would handle, then those tags won't be handled as long as there are old p-code in use. That is not a big problem, since p-code typically isn't generated from source on the frontend, but it will have the effect that tag modules that exist on the frontend and not on the (p-code generating) backend will have no effect.

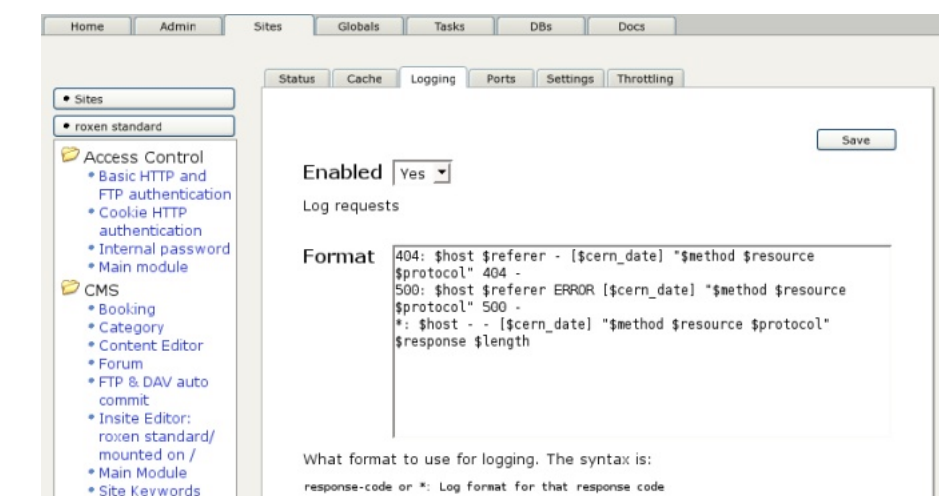
## 8.9 Cache Monitoring and Performance Analysis

### 8.9.1 Cache crawler status

This wizard shows how many files that are queued for crawling in the disk cache at the moment. Files are normally sorted by directory depth and then alphabetically to ensure that site start pages and top-level section pages get the highest priority.

The server will process chunks of several items at once meaning that it may not be the page listed at the very top that is currently being processed.

### 8.9.2 Logging of \$cache-status



The cache utilization can be logged for every request by using the \$cache-status field in the server log, configured under Site/Logging/Format in the Roxen WebServer administration interface. It expands to one or more of the following:

**protcache**

Cache hit in the low-level cache in the HTTP protocol module. This cache is very fast but works only for completely static content.

**xsltcache**

Cache hit in the XSLT cache.

**pcoderam**

Cache hit for RXML p-code in RAM.

**pcodedisk**

Cache hit for RXML p-code in the persistent cache.

**cachetag**

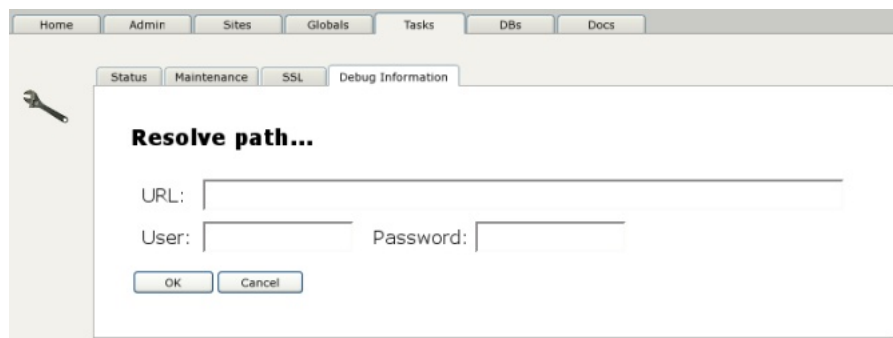
No cache misses in any <cache> tag.

**nocache**

No hit in any known cache.

This provides the raw data to be used e.g. by a log analysis tool.

**8.9.3 Resolve path**

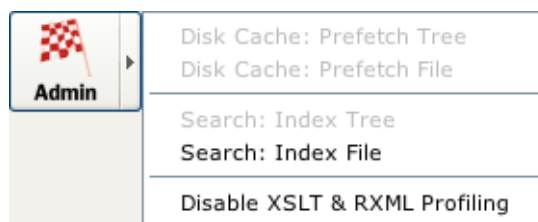


The request trace used by the "Resolve path..." action under Tasks/Debug information shows the cache hit/miss info and other details for every encountered <cache> tag in the page.

**8.9.4 XSLT and RXML profiling in Content Editor**

There is a user-friendly performance profiler in the Content Editor. It can be enabled by a single user for a period of time and then disabled again. While active it logs performance metrics for all pages that the user requests from his edit area.

To enable profiling, select it from the Admin menu under the Files tab inside the Content Editor. Its active status is indicated through a change of the menu icon:



At this point you should view your web pages again to collect profiling data. Only the most recent request to a given page will be recorded.

To view a profile for a particular page, focus on its file object in the Content Editor and scroll down a bit to see the full log.

The screenshot shows the XSLT Profiling tool interface. At the top, it says 'Profile captured: today, 21:46:31' with 'Clear' and 'Clear All' buttons. The main area displays a log of XSL transformations and tests, including file paths like '/roxen-files/cms-sites/4.5/css/index.css' and various XSLT instructions like <select>, <test>, and <call-template>.

```

82 ms XSLTransform: File: VCFile(N/E:2:[sb52]::/index.xml)?__sb_force_userid=2&1320679641, override XSL: 0
58 ms <xsl:template> @match: /
0 ms <xsl:variable> @name: has-form-calendar => empty node-set
0 ms <select: //form-text-component[variant = '3'] => empty node-set
0 ms <Link>
0 ms <select: /: $css-file => "/roxen-files/cms-sites/4.5/css/index.css"
0 ms <Link>
0 ms <select: /: $print-css-file => "/roxen-files/cms-sites/4.5/css/print.css"
0 ms <Link>
0 ms <select: /: $handheld-css-file => "/roxen-files/cms-sites/4.5/css/handheld.css"
0 ms <Link>
0 ms <select: /: $print-css-file => "/roxen-files/cms-sites/4.5/css/print.css"
0 ms <xsl:if>
0 ms <test: $has-form-calendar => FALSE
0 ms <xsl:if>
0 ms <test: //form2-component => FALSE
0 ms <xsl:if>
0 ms <test: //flash-video-component => FALSE
0 ms <xsl:if>
0 ms <test: //share-component[variant = 3] => FALSE
0 ms <xsl:if>
0 ms <test: //kaltura-list-component or //kaltura-video-component => FALSE
0 ms <xsl:call-template> @name: onload
0 ms <xsl:template> @name: onload
0 ms <xsl:if>
0 ms <test: $has-form-calendar => FALSE
15 ms <xsl:variable> @name: tree => tree fragment (root: OutputNode(<nav-nodes>))
15 ms <xsl:call-template> @name: nav-render-xml
15 ms <xsl:template> @name: nav-render-xml
0 ms <xsl:param> @name: page-path => node-set (1 elem: (#16000002, <path>))
0 ms <select: xml:metadata()/path => node-set (1 elem: (#16000002, <path>))
0 ms <xsl:param> @name: path => "/"

```

## 8.10 Multiserver Configuration

All settings are made on the backend server and automatically replicated to the frontend machines. The prefetch crawler only runs on the backend.

To understand what files are stored where, let's consider a conceptual server setup with one backend server and some frontend servers.

On the backend server, we will have one set of files per work area. Each workarea has edit areas for all users, and a view area. All cached rendered files reside in the view area.

On each frontend server, we only have the live view area, with cache files replicated from the corresponding workarea on the backend server.

When a content file is edited in Workarea: Main on the backend server, the prefetch crawler will create a matching cache file (unless any of the prefetch restrictions mentioned earlier prevents this, of course). Both these files, the content file and the cache file, will be replicated to each frontend server.

Since on-demand transformations can be cached persistently as well there may be cache files on the frontend servers which don't correspond to cache files on the backend. Even though there could be a replication transaction which only regenerates a subset of the cache files (due to e.g. maximum crawl time), the remaining obsolete cache files on the frontend will automatically be marked as invalid in order to preserve consistency.



# 9 Database Maintenance

---

On the Maintenance page these two actions are available.



## **Purge references to removed modules from the database**

This will remove all references to modules that are not successfully loaded at the moment. This includes protection points belonging to the modules, or authentication data used by missing authentication modules.

## **Dump database to file**

This will create a database dump in the SiteBuilder storage. This is useful to do when for example moving the storage to a new server installation. When starting a new site with an old storage, this file will be read and the contents transferred to MySQL in the new server.

# 10 Replication

---

The objective of the replication system is to provide distributed exact copies of the SiteBuilder view area in a CMS Advanced powered site. This promotes better access ratios for users and by having many frontends of a site running, prevents the site from crashing in case of too much load.

The replication system makes it possible to distribute a site globally by providing secure connections between the replication server (backend site) and the replication clients (frontend). The replication clients are a crippled version of a regular CMS Advanced site, since it does not contain a Content Editor where the files can be edited.

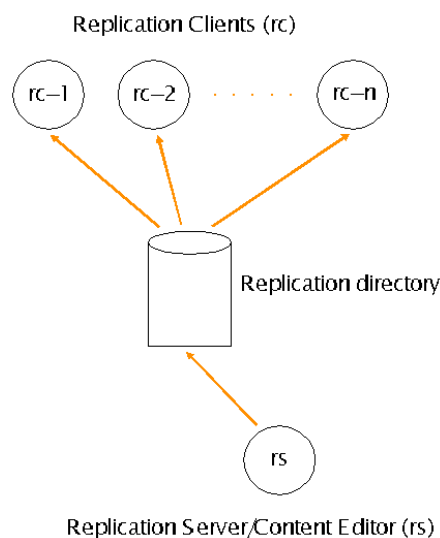
The replication server is used by the editors like any normal SiteBuilder site. The editors will only be affected by the replication system when it is time to distribute the changed documents to the replication directory (and limited tags, e.g. sb-edit).

The actual propagation of a site is done by using an intermediate directory (replication directory) where all necessary data from the replication server is stored. The replication clients poll the replication server for new information. The replication directory is a one-way communication channel from the replication server to the replication client. The replication clients have no other connections with the replication server.

## 10.1 Replication Methods

Currently, there are two allowed replication methods.

The first method, shared replication directory in a filesystem, should be used when both the replication server and the replication clients operate within the same file system. When utilizing this method, any number of replication clients may share one and only one replication directory in the file system, for example using a network file system like NFS or similar.



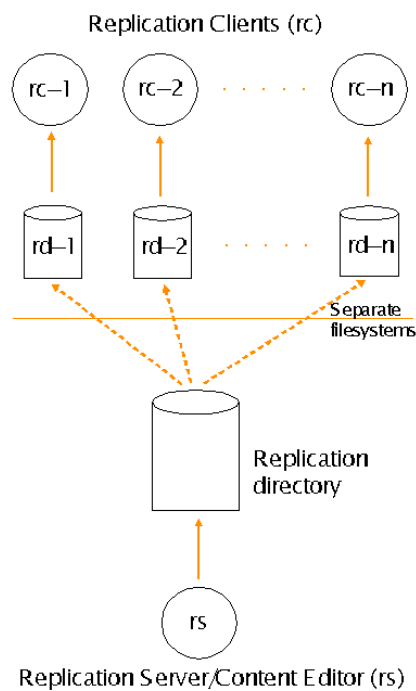
*Shared replication directory in a file system, e.g. NFS.*

The second method, distributed replication directories, should be used when the replication server and the replication clients use separate file systems. This method uses applications like scp, rsync or rdist over HTTP to copy the information from the replication server to the replication client file system via a script. The script is invoked by each replication client and sends a request for new files to be sent from the replication server to the replication client's own replication directory. The replication system tries to make sure that the script can copy the replication directory in a synchronized manner, where a) the replication server makes sure that the files in the replication directory can be copied at any moment and that b) the replication clients do not use the files during the time the script is copying them from the replication server to the replication directory.

The preferred method of distributing the information to the replication directory is rsync, since we have found it to be the most reliable application.

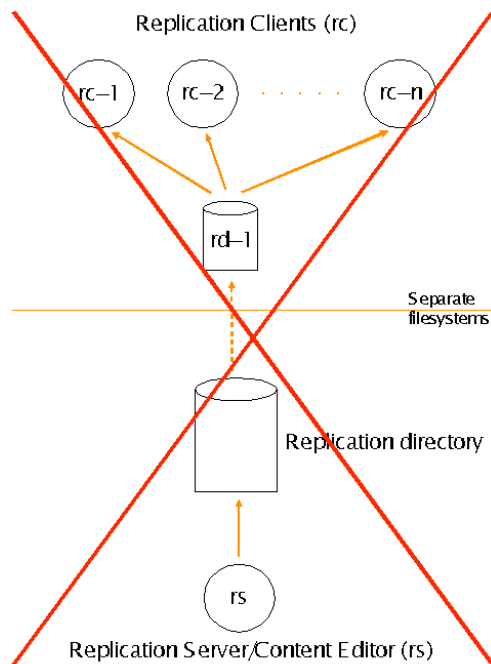
**Note!**

It is very important that each replication client has its own replication directory in this type of configuration. A faulty setup may cause severe synchronization problems when each replication client tries to fetch files from the replication server to the replication directory.



*Distributed replication directories by broadcasting via i.e. rsync, scp or rdist applications.*

The problem with this method is that during the time a replication client script is copying files from the replication server to the replication directory, the replication directory files are not consistent which means that other replication clients should not use them. Since the current synchronization system only is designed for a system where there only is a one-way communication channel from the replication server to the replication clients the other replication clients will not know that they are using inconsistent files from the replication directory.



*Do not set up the replication system like this!*

## 10.2 Replication Setup

This chapter might be of some help when setting up the replication system. The first part concerns the concept of dump files and difference files.

The replication directory contains two types of files created by the replication server:

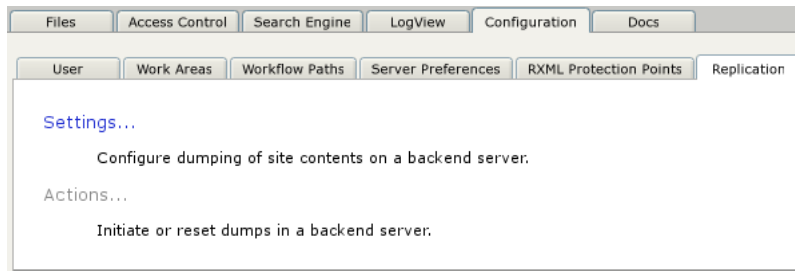
1. Dump files which contain a complete view area including Access Control database representing a consistent state.
2. Diff (difference) files which contain changes between two consistent states in the view area and changes in Access Control.

Dump files are named <sequence number>.dump and diff files are named <sequence number>.diff. The sequence number is a decimal number which is incremented by one for each diff file issued by the replication server. The replication clients keep track of the expected sequence numbers. The replication server and replication clients will automatically try to recover from unexpected sequences (e.g. sequences broken by a replication server crash, etc..).

Normally, the replication clients will only apply a dump file once and after that only apply diff files (regardless of new dump files created by the replication server). The replication clients poll for new dump and diff files regularly. A replication script (if available) is invoked prior to the poll to copy the contents of the replication directory if necessary. Old dump and diff files are removed automatically.

## 10.3 Setting up the server

Start setting up the replication system by configuring the replication server settings. Click on the Replication backend button Settings on the SiteBuilder Content Editor Configuration tab page. By clicking on the Actions button, a dump can be started.



### *Content Editor Configurations tab*

The settings for the replication server are as follows:

#### **Enable replication backend**

The Enable replication backend setting enables/disables this particular SiteBuilder server to function as a replication server.

#### **Replication workarea**

The Replication workarea setting is used to indicate the workarea to be replicated. Currently only one workarea may be replicated.

#### **Dump weekday and Dump hour**

How frequently the whole site should be dumped. Dumping the whole site, including the Access Control database, might take a lot of time. One dump per week should be enough for most sites.

#### **Replication path**

Sets the path to the replication directory where the server collects its diff (difference) files and dump files. The path can only be changed when Enable is set to "No".

Settings		
Variable	Setting	Help
Enable replication backend	Yes	Selects whether you would like to enable the replication backend.
Replicated workarea	Main	Selects which workarea you would like to replicate.
Dump automatically	Yes	<p>If dumps are made automatically then there is a complete replication state available on disk at all times to bootstrap or recover frontends. On the other hand, maintaining the dumps requires a great deal of server resources if the site is large.</p> <p>If dumps are not made automatically then that work is avoided, but a dump has to be triggered manually before a frontend can be bootstrapped or recovered, and that can delay recovery several hours.</p> <p>If dumps are made manually then "Once" can be used to schedule a single dump at off-duty hours according to the automatic dump schedule below. After that this setting automatically goes back to "No".</p>
Automatic dump weekday	Monday	Selects day of week to do automatic dumps if that is enabled.
Automatic dump hour	03	Selects hour of day to do automatic dumps if that is enabled.
Time to keep diffs and dumps	2 weeks	<p>Selects the time to keep diff and dump files. Files older than this are removed automatically, meaning that a frontend never may lag behind longer than this for it to keep up-to-date.</p> <p><i>Note:</i> This setting is only applicable when automatic dumps are disabled, otherwise the time to keep files are governed by the automatic dump frequency, so that a complete replication state is available at all times.</p>
Replication path	../www.roxen.com.rep/	Replication path.

OK Cancel

### Replication server settings wizard

#### 10.3.1 Setting up the client

When the correct settings for the replication server are complete, steer the browser to the Roxen Administrations interface. The replication client runs on a crippled SiteBuilder site, i.e. no Content Editor is available. This means a new site with a special site template has to be created.

New sites are created under the Sites. Give the replication client a proper name and create a new site with this template: SiteBuilder frontend site.

To finish, set the URL for the replication client frontend site and the initial variables for SiteBuilder:

#### Replication client poll cycle

The amount of time between polling the replication directory for new files and running the Replication client script (which - if available - polls the replication server for new difference and dump files). A poll can be issued by the system more often, when needed.

#### Replication client script

Sets the path to the optional replication script, which polls the replication server for new diff and dump files and copies them to the replication directory. This script is necessary if the replication client and the replication server do not share the same replication directory, i.e. the same file system.

#### 10.3.2 Setting up the replication directory

The last two settings of the initial SiteBuilder variables settings concern the replication directory.

## Replication directory

Which replication method is the site using? Has the replication client and the replication server the possibility to share the same replication directory, i.e. using the same file system? If this is true, then the replication system uses shared replication directory (replication over NFS, etc.), as discussed in the Replication section. Then specify the same replication directory information as used in the replication server settings. Otherwise, if the replication system uses distributed replication directories (distributed replication via script), set the path to where the Replication client script should copy the replication server difference and dump files.

## SiteBuilder storage

Sets the directory where SiteBuilder will store the site contents, fetched from the diff and dump files in the replication directory.

### Note!

The replication client and the replication server must not share the same SiteBuilder storage area. Make sure that the path does not point to an existing directory.

### Initial variables for the site

URLs  :80  Bind this port:  Yes  No

Bind to these URLs. You can use '\*' and '?' to perform globbing (using any of these will default to binding to all IP-numbers on your machine). If you specify a IP# in the field it will take precedence over the hostname.

### Initial variables for CMS: Main Module

SiteBuilder storage

The directory where SiteBuilder will store the site contents. Note that it's not possible to use several SiteBuilders with the same storage.

Please make sure it points to a good place. Unless you're importing an existing SiteBuilder site, it should not be an existing directory.

When changing this path in an already running instance of the module, you have to reload the module before the change takes effect.

Replication client script

This script is useful if the replication client and server do not share the same "Replication directory".

There is one possible type of command given as the first argument to the script: `copy`. For `copy`, the script should copy the contents of the replication server directory to the replication client directory.

Note: When using `rsync`, provide the `--delete` flag.

Replication directory

This directory can either be shared with the replication server directory, or be a copy of the server directory using the "Replication script".

Replication client poll cycle

The time between running the "Replicate script" (if available) and checking the "Replication directory". A poll can be issued by the system more often when needed.

### *Initial variables for replication client frontend site*

When the initial variables for the replication client has been set, there is one more setting left to set. The work area has to be the same in the replication client as set in the replication server settings. Click on SiteBuilder Work Area file system and choose the correct work area under the Settings.

# 11 Workflow

---

## 11.1 Concepts

The Workflow section in the Insite Editor manual contains an introduction to the concepts of the workflow module and a detailed description of the user interface for working with workflow-controlled files.

In addition to the concepts presented elsewhere an administrator should also be aware of the distinction between the following:

- Workflow processes and its activities
- Workflow process definitions and accompanying activity definitions

Regular users rarely need to be concerned with definitions since they only interact with files that are workflow-controlled. However, during administration of workflow processes the definitions play a central role: they are the blueprint from which the actual workflow processes are created.

For instance, let's assume your site contains a definition for a workflow process called News Workflow. When a news page is created, or when a change takes place for a page not already under workflow control, an important thing happens: the data contained in the workflow definition is used to initiate a workflow process and its activities. At this time the targeted page gets a brand new process cloned from the definition. The significance of this is apparent when someone decides to reassign or reject the page; at that point it's possible to select different recipients for that particular page without affecting any other page also controlled by News Workflow.

Another benefit is that the system applies version control to the definitions themselves. A definition can be changed without affecting processes that were cloned from an earlier revision since they continue to run according to the rules and settings that were valid when they started. The obvious consequence of this is that administrators cannot instantly alter the behavior of workflow definitions currently in use since they need to end and restart before they recognize the updated definition.

## 11.2 Definition files

Workflow process definitions are stored in XML files located in the site repository. A definition file can have any name as long as it is assigned the Workflow Definition content type. The process it defines is only active in the directory containing the definition file and its subdirectories, and only after the definition has been committed. A definition can furthermore contain restrictions on path and content type for the files it wants to control.

When looking for definition files Roxen CMS will employ its standard SmartSearch strategy which selects the one closest to the target file starting in the target file's directory and then continuing to its parent directory and its ancestors, ignoring any ineligible definitions in the process. If multiple candidates are found in the same directory the choice is currently made in alphabetical order. Definition files that have their Selectable metadata set to "No" will also be skipped in the search.

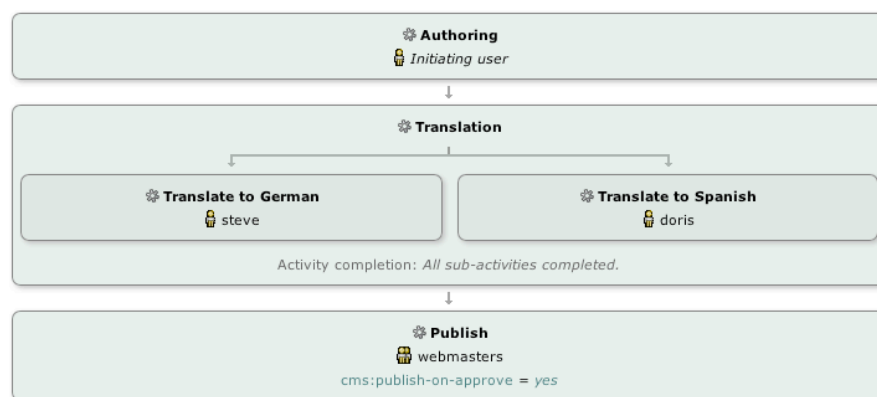
A special exception is made in the case of a workflow definition controlling itself. Please note that a definition file may however be handled by any other workflow process that covers that location.



The XML format can be seen as a textual representation of the graphical flow chart describing the process. All activities are placed inside a container, and additional nesting levels can be used to represent subactivities. This nesting is used to reflect that the enclosed subactivities need to complete before the parent activity is considered complete. Note that the parent activity in this case is purely a container and not an activity that users need to complete separately.

Each activity enclosing nested subactivities has a property that controls whether the subactivities can execute sequentially or in parallel. In the sequential case the subactivities are started one by one as soon as the preceding one is completed. The parallel model will start all subactivities simultaneously and then wait for one of two possible results: either one of the subactivities or all subactivities need to be complete depending on how the parallel activity is configured.

To illustrate with an example, consider the following definition:



This definition has one untitled top-level activity (not visualized but required in the XML markup) containing three subactivities executing in sequential fashion. The second activity is another container that holds two subactivities that run in parallel and it should be configured to require both of them to complete before the parent activity is complete since we need both German and Spanish translations in place for the publishing to be approved.

In other scenarios we can imagine parallel activities being used to represent either-or approval situations where at least one but not necessarily all recipients give approval. In such circumstances it can be worthwhile to use a workflow feature that makes the files read-only during these activities.

The definition should assign one or more AC user/group identities for each activity (except activities that only act as containers for subactivities). If only a single user is assigned that user will receive this activity directly when it starts; if multiple users or a group (or combinations thereof) are chosen the activity will be visible in the to-do lists of all relevant persons until one of them accepts the activity by clicking the Take button.

There is a special type of assignment that represents the AC user that initiated the whole workflow process. By including this assignment in one or more activities they will be decided when someone creates or edits a file. It's generally a good idea to have the first activity of the workflow process use this type of assignment.

Both the process definition and the activity definitions it contains may have one or more application-defined attributes. The workflow module is considered to be of general nature that may see uses for various purposes in the future. Presently, in the context of Roxen CMS there are a number of behaviors that need to be defined via

CMS-specific attributes relating to files, e-mails and versioning. In particular, one attribute controls whether approval of an activity triggers an external commit of the associated file in the repository.

Details of the XML syntax and relevant attributes can be found in the section below.

## 11.3 User interface

Aside from the end-user interface referenced earlier there are three additional interfaces that are relevant to administrators:

### The Workflow Status wizard

When working in the Content Editor this can be found in the “Files” tab under the “View” menu. It is identical to the Workflow Status wizard that can be accessed in the Insite Editor toolbar.

### The Status wizard

This wizard is located in the “Configuration” section of the Content Editor under the “Workflow” tab. It lists all workflow processes in use in the site together with a usage count and a link to the corresponding XML definition files.

### The workflow definition file handler

A file handler is responsible for viewing and editing files of a certain content type in a Roxen CMS repository. By assigning the Workflow Definition content type to an XML file you register it as a workflow process definition.

The handler parses the XML format and display a graphical preview of the process (see the picture in previous section) together with various options and parameters contained in the process definition. It also performs a syntax and consistency check on the definition to catch errors; in case of problems the definition cannot be committed and not become active until corrected.

## 11.4 Workflow XML syntax

### 11.4.1 Sample workflow definition

The previously used example can be translated into XML syntax in this way:

```
<?xml version="1.0"?>
<process-def>
  <name>News Workflow</name>

  <activity-def>
    <sequence-model>sequential</sequence-model>

    <activity-def>
      <name>Authoring</name>
      <assign></assign>
    </activity-def>

    <activity-def>
      <name>Translation</name>
      <sequence-model>parallel-all</sequence-model>

      <activity-def>
        <name>Translate to German</name>
        <assign>steve</assign>
      </activity-def>

      <activity-def>
        <name>Translate to Spanish</name>
```

```

        <assign>doris</assign>
    </activity-def>

</activity-def>

<activity-def>
    <name>Publish</name>
    <assign>webmasters</assign>
    <attr name="cms:publish-on-approve">yes</attr>
</activity-def>

</activity-def>
</process-def>

```

After entering this definition and saving it you will notice that Roxen CMS adds various ID attributes to both `<process-def>` and `<activity-def>` elements. This is necessary to uniquely reference the definition and its components if the user-visible names would change later.

### 11.4.2 Basic structure

The whole definition must be placed inside `<process-def>`. This element should contain (aside from name and attributes) a single `<activity-def>` that in turn encapsulates any number of `<activity-def>` children.

Each `<activity-def>` element except the top-most one needs a `<name>`. If there are nested activity definitions you must also determine the sequence model used: `sequential`, `parallel-one` with at least one completed activity, or `parallel-all` with all children completed.

Assignments for an activity definition are specified with one or more `<assign>` elements. You can provide the Access Control identity in either numerical or string form, though the latter requires the “handle” (i.e. the account name) and not the full name of the user or group. Please note that groups normally don’t get handles unless manually entered in the AC interface. An empty `<assign>` element indicates the Initiating User exception meaning that the workflow module will substitute the AC identity of the user that creates or edits the file.

### 11.4.3 Attributes

Attributes can be attached to both `<process-def>` and `<activity-def>` elements using `<attr>`.

Roxen CMS supports the following attributes:

#### **<process-def>**

```

    cms:content-type
    cms:path-include
    cms:path-exclude
    cms:activity-on-delete
    cms:email-on-approve
    cms:email-on-reassign
    cms:email-on-reject

```

#### **<activity-def>**

```

    cms:publish-on-approve
    cms:publish-on-commit
    cms:email-on-approve
    cms:email-on-reassign
    cms:email-on-reject
    cms:read-only

```

Usage of each attribute:

**cms:content-type**

Comma-separated list of glob patterns (i.e. wildcards \* and ? allowed) that decides which content types that the workflow definition applies to. The content type for a file can be seen in the “File Types” wizard.

Example:

```
<attr name="cms:content-type">text/css</attr>
```

**cms:path-include, cms:path-exclude**

Comma-separated lists of glob patterns that includes or excludes file paths with regards to workflow control. If neither is given all files located in the same directory as the definition itself or any of its subdirectories are eligible for workflow. If both include and exclude patterns are provided the exclusion has highest priority.

Example:

```
<attr name="cms:path-exclude">*_hires.jpg</attr>
```

**cms:activity-on-delete**

The ID value for the <activity-def> element corresponding to the activity that the process should transition to when a workflow-controlled page is deleted. This activity is normally intended to be the last activity in the process where an approval action terminates the process and completes the deletion through the use of the cms:publish-on-approve attribute.

If this attribute is not provided the workflow process will allow any user to delete pages without further approval.

Note that missing ID numbers are assigned automatically as soon as the workflow definition XML is committed.

Example:

```
<attr name="cms:activity-on-delete">4</attr>
```

**cms:email-on-approve, cms:email-on-reassign, cms:email-on-reject**

Each of these may contain a comma-separated list of e-mail addresses. The listed recipients will get e-mail messages when the corresponding action (approve, reassign or reject) has taken place.

Roxen CMS will use addresses declared both in the <activity-def> element for the current activity as well as addresses specified globally in <process-def> covering the whole process definition.

Aside from e-mail addresses explicitly listed here Roxen CMS will also use e-mails available in the Access Control system.

Example:

```
<attr name="cms:email-on-approve">doris@company.com</attr>
```

#### **cms:publish-on-approve**

If present this attribute triggers an externally visible commit (often referred to as “publish” to distinguish from hidden commits that aren’t seen outside of edit mode) when the activity is approved. It’s normally used in the final activity of a process definition but can be used elsewhere as well.

At least one activity in the definition must be flagged with this or the `cms:publish-on-commit` attribute, or all activities must be read-only for the process definition to be valid.

Example:

```
<attr name="cms:publish-on-approve">yes</attr>
```

#### **cms:publish-on-commit**

Alternative to `cms:publish-on-approve` that makes the committed file externally visible as soon as it is committed in its current activity. This means it can be published before the activity is approved.

Example:

```
<attr name="cms:publish-on-commit">yes</attr>
```

#### **cms:read-only**

A flag that prevents files associated with the activity to be modified. Useful for activities that request approval from someone without allowing the file to be changed.

Example:

```
<attr name="cms:read-only">yes</attr>
```

## 11.5 Important notes on implementation

All data concerning workflow is stored in the internal database. Even definitions are mirrored to the database to enable use of older revisions in parallel to current revisions of the same workflow process. This design allows for good performance in keeping to-do lists updated and enables operations on several files at once; however, the drawback is that workflow information is not maintained when exporting a site (either using the ZIP file or transport directory mechanisms).

It’s therefore important that the database is backed up regularly since the data contained inside cannot be reconstructed from the repository.

Prior to the introduction of this workflow module Roxen CMS had a different workflow solution that now is deprecated. Since the two modules are radically different in design and approach there is no migration path from the old version. Nevertheless, sites requiring the old module can be started with the `-DOLD_WORKFLOW` option to completely disable the new module and switch to the old behavior, though this option will disappear in future versions.

# 12 Link Management

---

## 12.1 Overview

### 12.1.1 Introduction to Link Management

Starting in Roxen CMS 5.0 the server offers built-in link management functionality. The purpose is to track and update links between files in the site repository regardless if the source and/or destination file is moved or renamed. Another benefit of the link tracking is that the system can warn about deletion of files still referenced from other places.

Tracking is currently enabled for the following page editor components:

- Text & Picture – Body text links, picture path and picture link.
- Link – Web link.
- Link List – Web links.
- Menu Link – Web link.
- File – File path.
- Insert Component – File path.

This functionality is transparent to the user and requires no additional steps in creating content. However, there are some technical characteristics that web developers and administrators need to be aware of.

### 12.1.2 File UUIDs

To make link management work the server needs to replace old-style direct references with link IDs that are resolved using a database. The database is updated whenever a page is created or modified so the corresponding link ID always resolves to the correct destination. In order to simplify the architecture the link IDs are actually not assigned per link but instead represented as a unique ID of a destination file, commonly referred to as a UUID (Universally Unique Identifier).

Roxen CMS will assign a UUID to a file as needed. UUIDs are only guaranteed to be unique within a single repository, and during editing a UUID may resolve to different paths for different users. Please see the “Implementation notes” later on for details. There are other internal modules that make use of UUIDs to track files (the Workflow module for example) so files can get UUIDs even if they aren’t linked to.

### 12.1.3 Link format

The UUIDs mentioned above are printed as a string of hexadecimal digits on a standardized format (lowercase with separators in fixed places). Here are some examples:

```
45212027-8c1b-421f-ba1e-5a0e302a7054  
a6a62322-cc19-451b-b3a0-d1d618111684  
18904ef9-483d-4186-a4bc-715fd461ccb7
```

When incorporated into a web link Roxen CMS will package them as follows:

```
/__uuid/45212027-8c1b-421f-ba1e-5a0e302a7054/index.xml  
/__uuid/a6a62322-cc19-451b-b3a0-d1d618111684/logo.gif  
/__uuid/18904ef9-483d-4186-a4bc-715fd461ccb7/house.jpg
```

The `/__uuid/` prefix makes it easy for the server to detect that a UUID follows so the rest is not interpreted as a regular path. Roxen CMS also implements an internal file system mountpoint for `/__uuid/` which understands this format so if you expose these links externally they will resolve correctly through an internal redirect.

At the end of the link you will normally see the current filename of the destination file. This part is not significant to resolving the file UUID but acts as a helpful reminder to developers that inspect XML source code. Directories can also have UUIDs and in that case the subsequent path elements will be processed relative to the target directory. The link URL can contain a query part as well starting with a “?” character.

If you have a UUID and want to know what file it points to, simply go to the Content Editor using an URL like this:

```
/edit/45212027-8c1b-421f-ba1e-5a0e302a7054
```

(If your CE interface is mounted on something other than `/edit/` you should substitute that part of the address.)

#### 12.1.4 Compatibility

There are some conversion steps required when migrating an old site to a modern Roxen CMS using link management. The system will enable link management automatically for newly created sites but leave it off for older sites until the administrator decides that it can be turned on.

The `/__uuid/` file system mountpoint will catch normal uses of UUIDs (e.g. when constructing `<a href="/__uuid/.../">` or `` tags) but if your code depends on seeing the actual path you need to explicitly resolve links during execution of your code. It's considered good style to resolve UUIDs server-side as much as possible to avoid exposing UUIDs to web visitors.

Similarly, developers of custom modules such as page component modules may need to add calls to create and resolve UUIDs during page editing.

In summary the following actions are needed:

- Update XSLT templates
- Update RXML code
- Update custom server modules
- Optionally force UUID assignment to existing files and convert old-style links to UUID format

The first two are covered in the “Link Management” section in the Roxen CMS Web Developer manual, the third in the “Editor Components” chapter of the System Developer (Pike) manual, and the fourth is detailed in the “Migration Strategy” section below.

## 12.2 User interface

You can find documentation on the interface presented in the sidebar panel and other end-user wizards in the “Link Management” section of the Roxen CMS Insite Editor manual.

### 12.2.1 Administrator settings

You find the administrator settings in the “Configuration” tab followed by the “Server Preferences” tab in the Content Editor. Click “Link Management...” to show the settings wizard with the following option:

#### Enable Link Management

Determines whether the Roxen CMS page editor should generate links with UUIDs instead of traditional file paths.

Please see the “Compatibility” and “Migration Strategy” sections for additional details on enabling this for older sites. Also note that changing this option will not affect existing pages, only future editing actions.

### 12.2.2 Reassigning Unique Identifiers

The “Reassign Unique Identifier...” wizard in the “Files” tab of the Content Editor lets you see the current UUID for a file or directory and optionally force generation of a new UUID.

In rare situations you can end up with conflicting UUIDs. It’s typically related to moves/copies or when reverting to older file versions, and the system will warn you by not allowing commits for affected files. You can then force a new UUID by selecting this command.

### 12.2.3 Export and Import wizards

Administrators can use the “Export...” and “Import...” wizards in the “Files” tab to move content within or between site repositories. The UUIDs associated with each file will be recorded in the external data during export. However, the actual content of the files will not be processed so links will be left untouched regardless if they are in UUID format or stored as file paths.

Next, when importing a previously exported archive you will see an option titled “Assign unique file identifiers (UUIDs) from imported files”. Checking or clearing this checkbox will have the following result:

#### “Assign...” checked

Roxen CMS will preserve the UUIDs of imported files which means that links between imported pages should remain intact. However, if importing to the same site that you exported from you will get a warning concerning UUID conflicts that prevents the import from running. If needed you can visit the existing pages and reassign their UUIDs manually as described in the previous section.

#### “Assign...” unchecked

Imported files will be given new UUIDs so the risk for duplicates is eliminated. The obvious disadvantage is that any links between files in the imported data will no longer resolve correctly.

### 12.2.4 Administrator interface actions

To see administrator actions related to UUIDs, navigate to the “Link Management” module and click its “Status” tab in the administration interface:

#### Assign missing UUIDs

This button will crawl through the site and assign new UUIDs to any file or directory that hasn’t got one already. It’s a one-time operation for sites migrating from Roxen CMS older than version 5.0. Please note that this operation can be time-consuming for a large site.



## Reindex

Rebuilds the internal UUID database from metadata stored in the site repository. Should only be necessary if the repository was moved into an existing server installation or if the database is corrupt.

## Stop indexer

Stops either the assignment or reindex operation started by the aforementioned actions.

## 12.3 Migration strategy

To activate link management in an existing site you need to ensure its templates are updated to handle the new UUID-based links. After that is done you can optionally convert existing links to take advantage of the location-independent references; if not done links will be converted to the new format only after editing of a particular page component.

This section covers the conversion process for content pages. The changes related to XSLT, RXML and component editor module APIs are documented in the “Link Management” section in Roxen CMS Web Developer manual.

### 12.3.1 Step-by-step guide

- Update templates and other custom server modules.
- Force UUID assignment to all files. Use the administration interface action to accomplish this.
- Export content to an external archive. Data is stored in the transport directory on your server. Although you have the option to download the archive in ZIP format to your local computer you will need the server’s Pike language environment to be able to run the script in the next step.
- Run an external Link Manager processing script. The script is found in the following location:

```
<path-to-roxen>/server-  
x.y.z/modules/sitebuilder/bin/link-mgr.pike
```

To invoke it you need to launch it with the Pike run-time and pass the location of the exported data as an argument:

```
<path-to-roxen>/server-x.y.z/bin/roxen \  
<path-to-script> \  
<path-to-exported-site>
```

The script will crawl through all XML files and substitute UUID-based links for the page components mentioned in the “Overview” section earlier.

- Import pages back into the site.
- Enable Link Management in the Configuration tab.
- Commit the changed files.

## 12.4 Implementation notes

Although normal operation of the Link Management module is easy to understand there are several complex situations that can arise. The reason for this is the staging

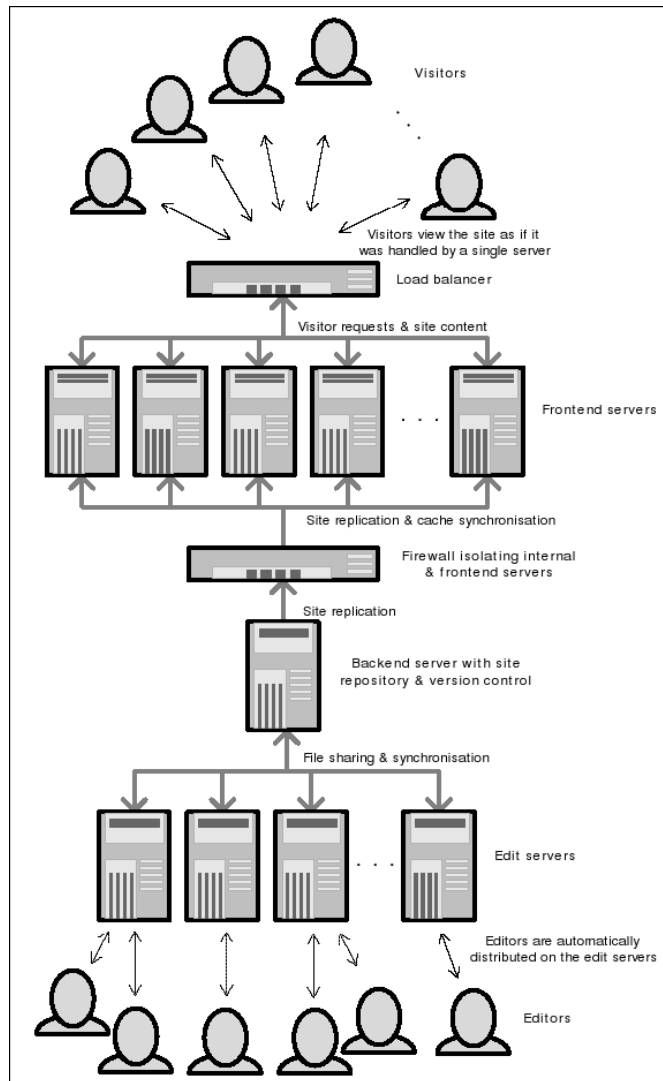
taking place in each user's edit area as well as the nature of file copying/moving operations.

The fundamental concept is that all files get a unique identifier when created and that this identifier should persist no matter where the file is moved. Now, let's consider a move operation in the Content Editor environment which technically is implemented as a copy plus a subsequent delete of the original file. Those operations are, however, only carried out in the user's edit area until committed. The result is that the UUID for the moved file will resolve to different locations depending on whether access happens inside or outside the edit area.

One tricky situation occurs if the pending delete of the original file is discarded in the edit area. This causes two files to have the same UUID in different locations which clearly is an unwanted state. Roxen CMS will react by blocking commit on the destination file in order to maintain a consistent UUID database and by letting the edit area UUID take priority over the repository UUID for edit area accesses.

Similar things can happen during operations such as copy, revert or upload, resulting in what can appear as unintuitive UUID handling for the sake of internal consistency. Commits controlled by workflow processes are another source of complexity.

# 13 Multiserver



*Multiserver example.*

## 13.1 Acceleration Frontends

An acceleration frontend is in essence a server with no edit mode. The acceleration frontend serves visitors to the site and does not handle editors. The purpose of an acceleration frontend is to remove visitor request load to the backend server. If the site is a high traffic site one could consider adding several acceleration frontends in combination with a load balancer to even out the load on the acceleration front end servers.

### 13.1.1 Multiple Edit Servers (MES)

A MES environment is preferable in case of large numbers of concurrent editors. A typical MES environment could consist of a frontend server, whose purpose is to serve pages to visitors, a backend server with the main site repository and one or

more edit frontend servers. The purpose of the edit frontends are to offload the main backend from the editor users work.

### 13.1.2 Acceleration Server Setup

In this section we will consider the details when installing a multi server setup. First, we'll typically want to share the Roxen Search database among the backend and frontend servers. The backend has write permission to the database, whereas frontends just have read permission to perform queries. To do this, you first need an external MySQL server that both backend and frontends can connect to, where Roxen Search can put its databases.

### 13.1.3 Backend Server

Before creating the backend site, go to the DBs tab in the Administration interface. Add a new group with the id "search", and optionally the description "Search Engine databases". The group's MySQL URL will be prefixed to any databases in this group. An example URL is "dbuser@dbserver.foo.com/". When creating a new site after this, its search databases will be created in the externally running MySQL.

Now, it is time to create your site, using the platform template as before, but this time setting the "Enable replication backend" setting to "Yes", making your site the backend server from SiteBuilder's perspective. See the persistent cache chapter and the replication part of the SiteBuilder chapter for further details. After the setup has finished, make an initial replication dump manually by clicking the "Dump" button under the Configurations tab's Replication Backend Actions in the content editor.

### 13.1.4 Acceleration Server

An Acceleration Server is a frontend server with the task of serving pages to site visitors, relieving the backend server(s) of this task. You can have multiple loadbalanced Acceleration Servers to improve access times and reliability of the site at high loads. To set up an Acceleration Server, again create the "search" database group (at the server in question, of course), pointing it at the same MySQL server URL prefix. Import the databases the backend site created here (names depend on what you called your site; e.g search\_1, search\_demolabs).

While at the DBs tab, create a database with the name "replicate", which will be used to share Roxen WebServer's argument cache among all servers of your setup. This is necessary to ensure that all servers, for instance, can show images generated by `<gtext>` regardless of which server that generated the image the first time. The argcache replication is symmetric, hence no server is more backend than any other in this respect. Under the Globals tab, Settings subtab, set the "Enable replication system" option to "Yes" to enable the argument cache sharing. You will have to restart the server before this is actually put into effect. Read more about the argument cache system in the argument cache section.

Then create a new site with the "SiteBuilder frontend" site template. Set up the rest of the replication syncing options as described in the replication part of the SiteBuilder chapter. Finally, set the Database setting of the module Search: Query Module to the database named after the site at the backend server, for instance search\_demolabs.

## 13.2 Argument Cache

The argument cache system keeps track of the mapping between dynamic generated id:s and their tag arguments and content.

When a request arrives for a page with a <gtext> tag the RXML parser generates a unique key and stores the arguments and the contents of the tag in the argument cache. The parser also replaces the tag with a <img> tag pointing to an internal image with the key as name.

The browser will later send a request for the image. When the request arrives the server extracts the key from the image name and asks the argument cache for the data stored for that key. The server will then generate and return the requested image.

### 13.2.1 Replicated argument cache

A shared argument cache is only necessary if a multi server setup serves pages with dynamic graphic tags, like gtext and cimg.

There are mainly two reasons why it is important to share the argument cache between Roxen servers:

- More than one frontend server is used and the load balancing system in front of the servers randomly selects a server for a particular user for each request.
- At least one frontend server is used and graphical tags are pregenerated by the persistent cache crawler on the backend server.

In both cases it is important that an argument cache key generated on one server is valid on all servers.

The argument cache is replicated by two methods, from the backend to the frontends and between the frontends:

1. The first replication method, from backend to frontend, uses the same replication system as the normal SiteBuilder replication to transfer information. When a sitebuilder replication package arrives to the frontend the local arg cache database is updated as well as the remote "replicate" database.

If argument cache replication is enabled on the frontends the replication from backend to frontend is automatic. A SiteBuilder replication dump on the backend will transfer all entries in the argument cache on the backend. And a "Admin/Disk cache: Prefetch" operation will transfer all arg cache entries accessed by the crawler.

2. The other replication method, between frontends, uses a shared mysql database to exchange information. When an graphical tag is parsed the arg cache is updated in the local database and in the replicate database. When an image is requested to a server generated elsewhere the server first look in the local database and then in the replicate database for information for the given key. If the information is found in the remote database it is stored in the local database and the image is returned.

If the shared database goes down, locally cached images will still be served but new images will not be replicated until the database goes online again.

### 13.2.2 Configuration

To configure a shared argument cache environment follow the steps below:

- The "Enable Replicate System" variable should be enabled on all SiteBuilder replicate frontends. In the Administration Interface enter "Globals" tab and "Settings" subtab select "yes" for "Enable replication system".

- A replicate database is required on all replicate frontends. In the DBs tab create a pointer named "replicate" to an external shared mysql database.
- The "Enable Replicate System" variable should be disabled on the SiteBuilder replicate backend. In the Administration Interface enter "Globals" tab and "Settings" subtab select "no" for "Enable replication system".
- The replicate database on the backend can be removed.
- Provided it is not important to have exactly the same image id:s on all frontends, for example when a "smart" loadbalancer is used, the replicate database could be internal and thus not shared between the frontends.

## 13.3 Multi Edit Server

A substantial part of the load on the edit server in a single edit server setup is consumed in the dynamic generation of the pages in the Content Editor as well as generation of previews of dynamic pages which users are working with. The actual Sitebuilder operations triggered by the clicking often takes less than half the time of the entire request processing.

The actual Sitebuilder operations require a certain amount of locking and synchronization which is not trivial to distribute over several machines, but the generation of Content Editor pages and user pages do not have such restrictions, and are quite possible to distribute. This is one of the two basic ideas underlying the edit server load balancing system described here.

The second idea is based on the following observation: some of the Sitebuilder operations, specifically the ones that only operate in a user's edit area, can be distributed with little overhead if the system ensures that the same edit area (and thus the same user) is almost always handled by the same server that handled the previous request for that user; in each situation where a different server is going to handle a user's request, the server that previously handled edit area operations for the user must empty all its caches related to that user's edit area.

### 13.3.1 Overview of the design

Basic structure: The system uses one Backend Edit Server, and one or more Frontend Edit Servers that handle Content Editor page generation, user pages plus those Sitebuilder operations that can be distributed without too much overhead. Other Sitebuilder operations, and non-Sitebuilder operations, is still handled by the Backend Edit Server, either by redirecting the user's request there, or by requesting through a HTTP RPC mechanism that the Backend Edit Server should perform the operation while otherwise still behaving as if the Frontend Edit Server performed the operation.

The Content Editor is adjusted so that the "Files" and "Docs" tabs are handled almost entirely by Frontend Edit Servers, while the other tabs are typically handled by the Backend Edit Server.

All the Frontend Edit Servers access the same CVS repository, the same view area, and the same edit areas, as the Backend Edit Server. In other words, a shared filesystem such as NFS will be required unless the Backend Edit Server and all Frontend Edit Servers runs on the same machine. It is viable to run all these servers on the same machine if the number of Frontend Edit Servers is reasonably small and the machine has multiple processors and enough memory.

### **Note!**

If NFS is used, it's extremely important that all forms of data and attribute caching are disabled. This behavior is usually achieved with the "noac" option when mounting NFS filesystems. The "noac" option is added to the client side mount.

### **Access Control**

The Access Control functionality is implemented by letting the Backend Edit Server and all Frontend Edit Servers share the same MySQL database. Only the Backend Edit Server is allowed to make changes to the database. Notifications of changes are propagated through the HTTP RPC mechanism so the servers won't have to throw away their caches at every AC change.

### **Load balancing**

The principle for distributing users between Frontend Edit Servers is implemented as a plain user ID modulo operation, statically dividing the users more or less equally between the Frontend Edit Servers. At the moment, any change in the number of Frontend Edit Servers on-line will cause automatic whole or partial cache flushes to ensure that each user is handled by one and only one Frontend Edit Server at the "same time". Moving a user between servers may lead to loss of non-persistent data such as variables in wizards and more importantly, when editing contents which have not been saved. It is therefore important to avoid changing the setup or restarting the Backend Edit Server (which causes a temporary restructuring of the setup while the Frontend Edit Servers reconnect which typically requires less than a minute after the Backend Edit Server has been restarted) while users are editing un-saved data.

### **Communication between servers**

The communication between the servers is bidirectional, with both synchronous and asynchronous HTTP RPC connections from the Frontend Edit Servers to the Backend Edit Server, and asynchronous connections from the Backend Edit Server to the Frontend Edit Servers.

The handling of the network topology (specifically how the Backend Edit Server knows how many active Frontend Edit Servers there are, and how the Frontend Edit Servers know what server is the Backend Edit server) is resolved at run-time when the Frontend Edit Servers connect to the Backend Edit Server using the HTTP RPC mechanism.

#### **13.3.2 Notes about usage**

A Multiple Editor Server system is in many aspects identical to a single server system from a user's perspective. There is, however, one difference:

- Users may have to login more than once in the Content Editor. This is a consequence of that fact that many servers (editor backend and frontends) are involved and that the user is, if trying to login on the wrong server, redirected to the proper server automatically.

## **13.4 Multi Edit Server Installation**

### **13.4.1 Base installation and configuration**

A Multiple Editor Server system consists of one Editor Backend and one or several Editor Frontends. Each server is installed as a separate Roxen Platform server. It's convenient to install the Editor Backend before any Editor Frontends.

### 13.4.2 Editor Backend configuration

Steps to configure the Multiple Editor Backend:

1. Install a site using the "Platform site" template.
2. Select the "Settings" tab for the SiteBuilder Main Module.
3. Set "Multiple Editor Server Mode" to "Content Editor Backend".
4. The "Multiple Editor Server Shared Secret" must be equal on all editor servers, both backend and frontends, involved in the same configuration.
5. Click on "Save".
6. Select the "Status" tab and click on "Reload".

### 13.4.3 Editor Frontend configuration

Steps to configure each individual Multiple Editor Frontend:

1. Install a site using the "SiteBuilder Content Editor Frontend site" template.
2. Go to the section called "Initial variables for SiteBuilder: Main module" during the template installation.
3. Set "SiteBuilder storage" to the same value as for the Editor Backend.
4. Set the "Multiple Editor Server Shared Secret" to the same value as for the Editor Backend.
5. Set the "Multiple Editor Server Backend URL" to the URL of the editor backend SiteBuilder URL (typically something like <http://backend.roxen.com/edit/>).
6. Set the "Multiple Editor Server Frontend URL" to the URL of the editor frontend SiteBuilder URL itself (typically something like <http://frontend-1.roxen.com/edit/>).
7. Click on the "Ok" button and proceed with the installation.

### 13.4.4 Access Control (AC) configuration

The most common Multiple Editor Server setup involves a number of separate computers with one Roxen server on each. In this case an external MySQL database is required to hold the AC database.

Steps to configure the Access Control system:

1. Go to the DBs tab on the Multiple Editor Backend server.
2. Click on "Create new database".
3. Give it a name (like ac\_shared).
4. Select Type: "External".
5. Set the URL to point at the external MySQL with a database name (ex. mysql://my-sql-server/ac).
6. Click "OK".
7. Give your Roxen site WRITE access to the newly created database.
8. Go to the "Settings" tab for Access Control Main module in the Platform site.
9. Change AC Database to the external one. (ac\_shared)
10. Select the "Status" tab and and click on "Reload".



11. Repeat the steps 1-10 above for each frontend server with one important difference. At step 7 you give the frontend sites READ access.

When this is done the internal AC databases can be removed (ac\_sitename), they are no longer used.

An alternative setup is to use a multi CPU computer and have all Roxen servers on the same machine. In this case it is possible to use one of the servers internal MySQL server for sharing AC database. The setup procedure is the same as above, but the MySQL URL is modified to point at one of the servers MySQL socket in the filesystem. The URL should look like this:

```
mysql://rw@localhost:/path/to/roxen/configurations/_mysql/socket/ac_shared
```

### 13.4.5 Search configuration

#### Diagnostics

The "Status" tab for the SiteBuilder Main Module contains some useful diagnostics for both the Editor Backend and the Editor Frontends:

- On the Editor Backend, each enabled Editor Frontend is listed next to the "Connected Multiple Editor Frontends" header.
- On each Editor Frontend, the Editor Backend is listed next to the "Connected Multiple Editor Backend" header.
- The "requested N s ago" information indicates when a remote server (editor backend or frontend respectively) was last heard of on each server. These values should typically be below 120 seconds which is the default internal ping cycle of the system.

## 13.5 Test Server Setup

A Test Server Setup involves two servers. One server is the Live Edit Server and the other server is the Test Edit Server. There are two types of scripts to synchronize the contents between the two servers. The details are described below.

There are several valid purposes to use this kind of setup. First, it is safer to test some aspects of new versions of critical components on a completely different server than the Live Server. Second, it can load balance the editor environment and enable load, performance testing etc. without disturbing the Live Server.

### 13.5.1 Live to Test copy

Copying from the Live to the Edit server means overwriting everything in the SiteBuilder Test Storage with the contents of the SiteBuilder Live Storage. Thus synchronizing the site contents completely except for optional databases (see below).

Use the script located in roxen/server-\*/modules/sitebuilder/bin/sb-sync.sh to copy site contents from the live server to the edit server. The script is designed to copy the SiteBuilder Storage directory only. To extend this and copy other directories, like the MySQL databases, the recommended solution is to create a local script in the roxen/local/bin directory which invokes sb-sync.sh.

Script arguments:

```
sb-sync.sh <SiteBuilder Storage src dir><SiteBuilder Storage dest dir>
```

## 13.5.2 Test to Live copy

Copying from the Test to the Live server is different from the other case because only the difference in the View Areas between the Test and the Live site will be copied across the servers. In addition, the contents which differs is copied to a selectable Edit Area on the Live Server. Contents that should be copied must be committed on the Test Server before the script is run (see below).

This scheme makes it possible to adjust and do final tests on the Live Server, before the content actually is published. It's also possible to resolve any conflicts that may be caused by any same file that has been updated on Live Server meanwhile.

Use the script located in `roxen/server-*/modules/sitebuilder/bin/va-export-new-to-ea.sh` to copy site content difference from the Test Server to the Live Server. The script is designed to copy the View Area in the SiteBuilder Storage directory only, thus all files that should be part of the transfer must be committed on the Test Server. To extend the script and copy other files and directories outside the View Area like MySQL databases, the recommended solution is to create a local script in the `roxen/local/bin` directory which invokes `va-export-new-to-ea.sh`.

Script arguments:

```
va-export-new-to-ea.sh <SiteBuilder src dir><SiteBuilder Edit Area dest dir>
```

### Requirements

- The View Area on the destination SiteBuilder dir is assumed to be in a complete (normal) state.
- The Edit Area on the destination SiteBuilder dir is assumed to be created. Please create it otherwise. Example:

```
mkdir Site.sb/wa/view/2
```

This creates an Edit Area for the administrator (who has Access Control id 2).

- The destination SiteBuilder must be restarted after the import has been completed. It is not strictly necessary to shut down the SiteBuilders while copying, provided nobody is committing any files at the same time.

### Options

To exclude specific paths in the View Area from being checked, set the `EXCLUDE_PATH` environment variable to a `grep` expression matching these paths.

### Shortcomings

New directories on the source SiteBuilder are not created on the destination SiteBuilder. Files in new directories will not be copied. Solve this shortcoming by creating these directories on the destination SiteBuilder manually before executing this script.

### Note!

To avoid excessive conflicts with older versions, it is a good idea to use Discard Changes on the files for the Edit Area used.