FirstSpirit™
Unlock Your Content

FirstSpirit™ Security
FirstSpirit™ Version 5.0

Version	**1.4**
Status	**RELEASED**
Date	**2012-10-30**
Department	FS-Core
Copyright	2012 e-Spirit AG
File name	SECU50EN_FirstSpirit_Security

e-Spirit AG
Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

info@e-spirit.com
www.e-spirit.com

e-Spirit

Table of contents

1 Introduction

During generation, information – for each page reference in the Site Store and for each medium in the Media Store – is deposited in a local database, the so-called FirstSpirit™ Access Control database. This database is used to provide information about FirstSpirit™ objects, for example to provide access permissions which have been saved for an object. Synchronisation of the Access Control database with the currently released project status takes place automatically when the content is generated.

The FirstSpirit™ Security module is closely linked with the personalisation of content for the checking of access rights in the live system (so-called "user permissions" – cf. FirstSpirit™ Manual for Administrators). If, for example, an access right is to be realised for certain objects in live status, apart from the FirstSpirit™ Security module, the FirstSpirit™ DynamicPersonalization module is also required (which has to be purchased)[1].

Apart from the information from the Access Control database, the FirstSpirit™ Security module also provides a servlet for deployment of FirstSpirit™ content ("CRC Transfer Servlet"). The task of the servlet is to compare and adjust (synchronise) generated project files between the FirstSpirit™ server and the live system. The CRC checksum calculation can be used to determine changed files and only these files are updated. This differential upload accelerates the updating process in the live system.

This document is provided for information purposes only. e-Spirit may change the contents hereof without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. e-Spirit specifically disclaims any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. The technologies, functionality, services, and processes described herein are subject to change without notice.

[1] See FirstSpirit™ DynamicPersonalization module documentation

1.1 Overview of the functions

The FirstSpirit™ Security module supports the following aspects:

- Provision of information on individual FirstSpirit™ objects, e.g.:
  - the complete path to the object
  - the object's CRC-32 checksum
  - the date of the last change to the object
  - the date of the last update of an object's stored data
  - the object's unique identifier
  - the object's ID

  This information can be accessed via a script or a JSP page.
- Provision of access rights at object level:
  Permission information (user permissions and/or access rights) for visitors to the generated site (so-called "user permissions") can be deposited in FirstSpirit™ on many nodes in the project and are part of the meta information. User permissions can be maintained with the help of the permission component[2]. This information can be accessed via a script or a JSP page.
- Adjustment of the files on deployment (via a FirstSpirit™ servlet): If a FirstSpirit™ servlet is used for the deployment, the information provided can also be used to adjust the files (on the basis of the access control database). Changed, new and deleted files only are taken into account in the transfer. Unchanged files are not transferred again.

[2] For details of user permissions, see FirstSpirit Manual for Editors (JAVAclient), Chapter 13

1.2 Topics of this documentation

Chapter 1: A brief introduction to the concept and functional scope of the FirstSpirit™ access control database (from page 3).

Chapter 2: Describes installation of the "FirstSpirit™ Security" module on the server and installation of the web component in a project (from page 7).

Chapter 3: This chapter explains configuration of the "FirstSpirit™ Security" module and the web component (from page 10).

Chapter 4: A detailed description of all classes and methods which provide access to the access control database (from page 22).

Chapter 5: Describes how to de-/register and implement a listener via the CRC Transfer Servlet (from page 63).

1.3 Terms and concepts

1.3.1 Access control database and access rights

The data in the local access control database and the access control database in the web application is adjusted on the basis of the stored CRC-32 checksum and the path to a generated file.

An access control database consists of several "Access Control Lists" (ACL). These "access control lists" can consist of several files. A file in an ACL corresponds to a file generated during generation.

The central component of an ACL is the user permissions. Files with the same user permissions are collated in one ACL.

In FirstSpirit™ user permissions can be deposited on any node in the project and are part of the meta information. The permission information can be maintained with the help of the permission component.

Further information on the permission component is given in the "FirstSpirit™ Manual for Administrators" and the "FirstSpirit™ Manual for Editors (JavaClient)".

1.3.2 Deployment servlet ("CRC Transfer Servlet")

The task of the "FirstSpirit™ Deployment Servlet" is to compare and adjust (synchronise) generated project files between the FirstSpirit™ server and the live system. A checksum calculation is performed on the existing and the new generated files to be able to decide, on the basis of the checksum, which files have been changed (new, changed or deleted). (Comparison of the file creation date is insufficient due to asynchronous clocks in the different servers.) If differences are found by this checksum calculation, the new and changed files only are subsequently uploaded and/or outdated files are deleted (if no new file with the same name exists).

The differential upload therefore does not accelerate the generation time on the FirstSpirit™ server but the update process on the live system only.
As this update process can take a long time, especially in networks with a narrow bandwidth and in an extensive site, use of the FirstSpirit™ servlet enables the time required for the whole deployment process to be reduced[3].

The servlet is made available through the FirstSpirit™ Security module and can be adjusted to a specific project using the configuration dialog of the corresponding web application "FS Security WebApp" (see Chapter 3.1.3.1 page 13).

Furthermore, some classes and interfaces for registering and deregistering of listeners are available (see Chapter 5 page 63).

[3] For details of configuration of a deployment via a FirstSpirit™ Servlet, see Manual for Administrators

2 Installation

2.1.1 Installing the module on the server

The FirstSpirit™ Security module must first be installed within the server and project configuration application. To this end, the "Modules" menu entry is selected in the Server Properties area. Click the "Install" button to open a file selection dialog. The fsm file (fs-security.fsm) to be installed can be selected here. The successfully installed file is then displayed in the "Server Properties" dialog:

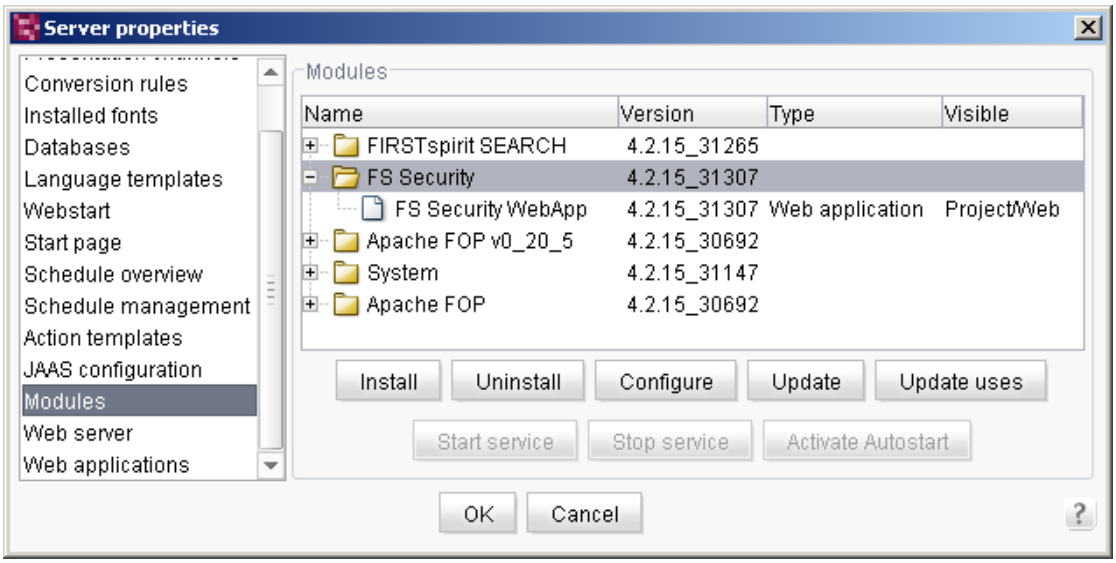

Figure 2-1: Installing the module on the FirstSpirit™ server

The web application "FS Security WebApp" is part of the FirstSpirit™ Security module. The web application provides the FirstSpirit™ publication servlet and the access control servlet.

“Scope” of the component are the "Project/Web" areas. It is therefore a "local web" component. After installation, this can be added to the different web areas (Preview, Staging, Live) within the required projects (see Chapter 2.1.2 page 8).

For further information on this dialog see "FirstSpirit™ Manual for Administrators".

2.1.2 Installing the web application in the project

The web application must now be installed in the required project. Open the "Web Components" menu entry within the project properties. The web components for a project can be activated in this area.

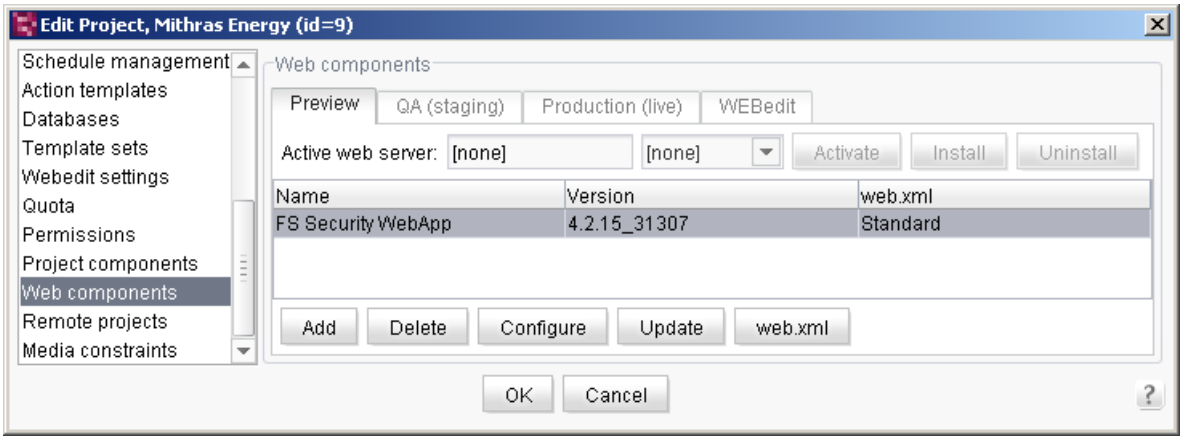

Figure 2-2: Installing the web application within the web areas

Three different web areas exist for each project. The respective tab can be used to individually activate and configure the web components for each area:

Figure 2-3: Web areas within a project

- Preview: Project contents location for which a preview has been requested.
- Staging: Location for the generated project content
- Production (Live): Location for the published project contents
- WEBedit: Location for WebEdit contents

Add: Click the button to open the "Add" dialog. The list displays all web components installed on the server (see Chapter 2 page 7).

After adding it to a web area it is possible to configure the component, either with a GUI generated by the component or a generic GUI (see Chapter 3.1.4 page 14). After configuring the components must be activated. A component within a project can be activated or deactivated for specific areas only

Install: Click this button to collate the web component in the project's respective web area in a War file and install it depending on the configured web server. The button is activated if the web

component has not yet been installed and is ready for installation. If the button is deactivated, the web component has already been installed.
If the selected web server is an external web server or a generic web server (without the necessary script function), the **"Download" button** is displayed instead.
If the web application has already been installed but the configuration has subsequently been changed, the **"Update" button** is displayed instead of the "Install" button (see Figure 3-4).

For further information on this dialog see "FirstSpirit™ Manual for Administrators".

2.1.3 System requirements

The deployment servlet is a servlet completely developed in Java and merely requires a servlet container (recommendations are given in the "Technical Data Sheet" of FirstSpirit™).

Note: The described servlets and servlet filters cannot be used in a cluster environment.

3 Configuration

The local access control database of a schedule is not created until the first generation and is updated with each further generation.

The data in a script is therefore available and usable within a schedule without further configuration. (For details on the use of scripts in schedules, see "FirstSpirit™ Manual for administrators".).

Further configuration steps are necessary only if:

- the information in the (local) access control database is to be used as the basis for updating, adding and deleting web server files (see Chapter 3.1.1 page 10).
- the information in the local access control database is to be transferred to an (external) web server (Chapter 3.1.2 page 11).

3.1.1 Adjusting the files through the access control database

File adjustment (synchronisation) on updating, deleting and adding files is based on the CRC-32 checksum and the path to a file. The path to a file stored in the local access control database is always absolutely related to the generation directory of a schedule.

Example: In German (abbreviation: "DE") the page reference "index.html" is generated for the first presentation channel.
The stored path is:

```
/de/index.html
```

For adjustment of the files, deployment via the "FirstSpirit™ Deployment Servlet" must be selected in a schedule. To do this a new schedule must first be added to the project's schedule management. The schedule is added to an action of the type "Deployment via a FIRSTspirit Deployment Servlet" (see documentation: "Manual for Administrators").

In addition, adjustments must be made in the web server. The crcTransfer.ini file is configured project-specifically through the configuration dialog of the web application FS Security WebApp (see Chapter 3.1.4.1 page 16). After the configuration the web application should be installed on the web server (see Chapter 2.1.2 page 8) and updated (see Chapter 3.1.4 page 14).

3.1.2 Transferring information to an external web server

Deployment via a FirstSpirit™ servlet frequently takes place in a subdirectory of the web application. The path to the root directory of the web application is therefore different to the generation.

On generation of a project a local access control database is therefore generated with all files. On deployment this local access control database is compared with and adjusted to (i.e. synchronised with) the remote access control database. The remote access control database can manage any number of projects. The absolute path is extended to include a project-wide, freely selectable prefix so that the individual files in the remote access control database can still be uniquely assigned. The given prefix supplements the path information of an object within the ACL database.

The prefix can be defined in the project settings in the "Access Control Database prefix" field in the "Permissions" sub-item:

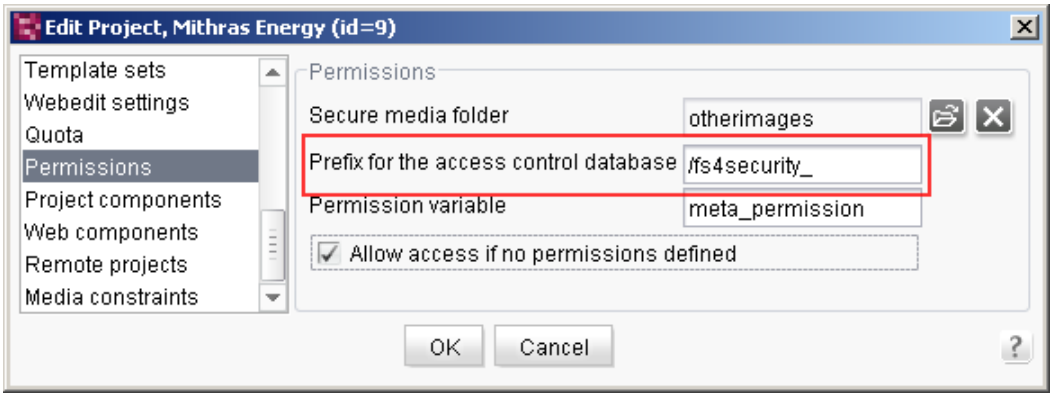

Figure 3-1: Entering the prefix for the file adjustment

Prefix for the access control database: Specify a prefix to complete the ACL database information of a file. The complete path to a file in FirstSpirit™ always consists of three parts:

- URL of the web application (e.g. http://myServer.de)
- Prefix for the Access Control database (/fs_security)
- Path to a file (/de/index.html)

The complete path within the ACL database is made up of the prefix and the path to the file, e.g. `/fs_security/de/index.html`

In Figure 3-1 the prefix is "/fs_security" and corresponds to the direct subdirectory "fs_security" of the web application, for example "live": "~Webserver/webapps/live/fs_security".

Specification of the prefix is absolute for the web application. In addition, in this case the prefix equals the last part of the "Path on live server" field value in the dialog of the deployment servlet (see Figure 3-2).

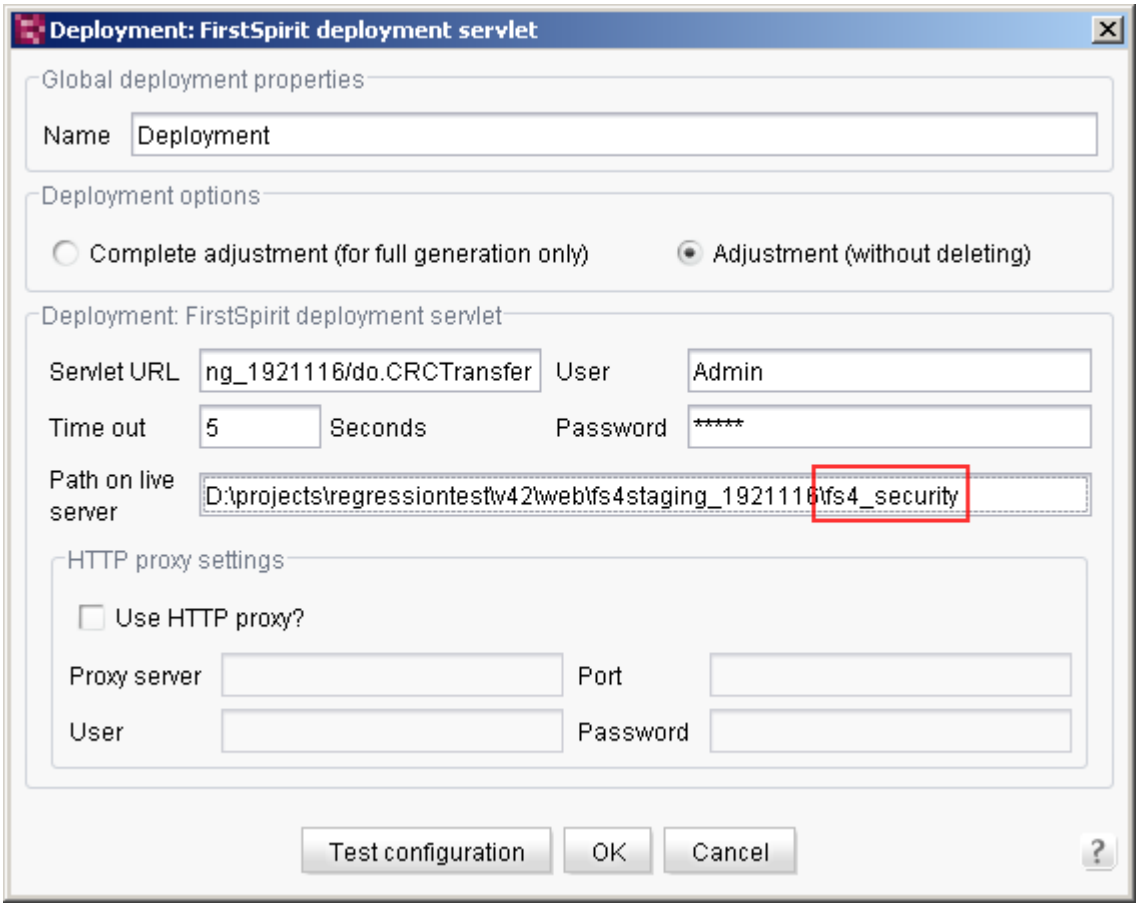

Figure 3-2: FirstSpirit™ Deployment Servlet dialog

The path of the "index.html" file from the example in the web application is:

```
/fs_security/de/index.html
```

If the prefix is changed, the entries are retained in the web application with the original prefix in the access control database.

A further important point is the "Adjustment (without deleting)" setting in the deployment servlet dialog. If this option is activated the entries for files which have been deleted in the project are not removed from the access control database.

3.1.3 Configuring the "FirstSpirit™ Security" module

The FirstSpirit™ Security module contains the following subareas:

- the FirstSpirit™ Deployment Servlet (see Chapter 3.1.3.1 page 13)
- adjustment of the access control database (see Chapter 3.1.3.2 page 14)

3.1.3.1 FirstSpirit™ Deployment Servlet

The deployment servlet is a fixed part of the web application in the web server. This servlet enables the local file list to be compared with and adjusted to the web application on deployment.

The CRC-32 checksum calculation for each file generated enables:

- changed and new files only to be transferred.
- Files no longer generated in the web application can be deleted (optional).

This ensures that all current data is generated, but only changed data is also transferred.

The data is adjusted on the basis of the calculated CRC-32 checksums. This checksum depends on the file content and file size. Therefore, information which changes on each generation, e.g. use of the generation starting time ("#global.startDate") within a meta tag for search engines, can have a disadvantageous effect on the deployment.

A servlet engine with full access to the file directory in which the website is located must be installed on the destination system to enable the FirstSpirit™ servlet to be used for deployment.

3.1.3.2 Access Control Servlet

The access control servlet is used to update the access control database on deployment. Here the status of the local access control database (via the servlet) is compared and adjusted to the status of the access control database in the web application.

Prerequisite for use of the access control servlet is use of the FirstSpirit™ deployment servlet.

Installation of the module is all that is required for use of the servlet (see 2.1.1 page 7). Alternatively, the servlet can also be manually inserted in the "web.xml" file of the web application:

```
<servlet>
  <servlet-name>
    fss-AccessControlServlet
  </servlet-name>

  <servlet-class>
    de.espirit.firstspirit.acl.servlet.AccessControlServlet
  </servlet-class>

  <load-on-startup>
    1
  </load-on-startup>
</servlet>
```

3.1.4 Configuring the web application

Click the button to open the dictionary configuration dialog (cf. Figure 2-2). The content of the two files "crcTransfer.ini" and "aclFilter.conf" is displayed in the configuration dialog (cf. Figure 3-3):

- The "crcTransfer.ini" file contains the configuration for the FirstSpirit™ deployment servlet (see Chapter 3.1.4.1 page 16).
- The "aclFilter.conf" file contains the configuration for the multi-access control filter. This file can be used to define different filters for project content for which special access protection is required within the generated and/or published content (see Chapter 3.1.4.2 page 18).

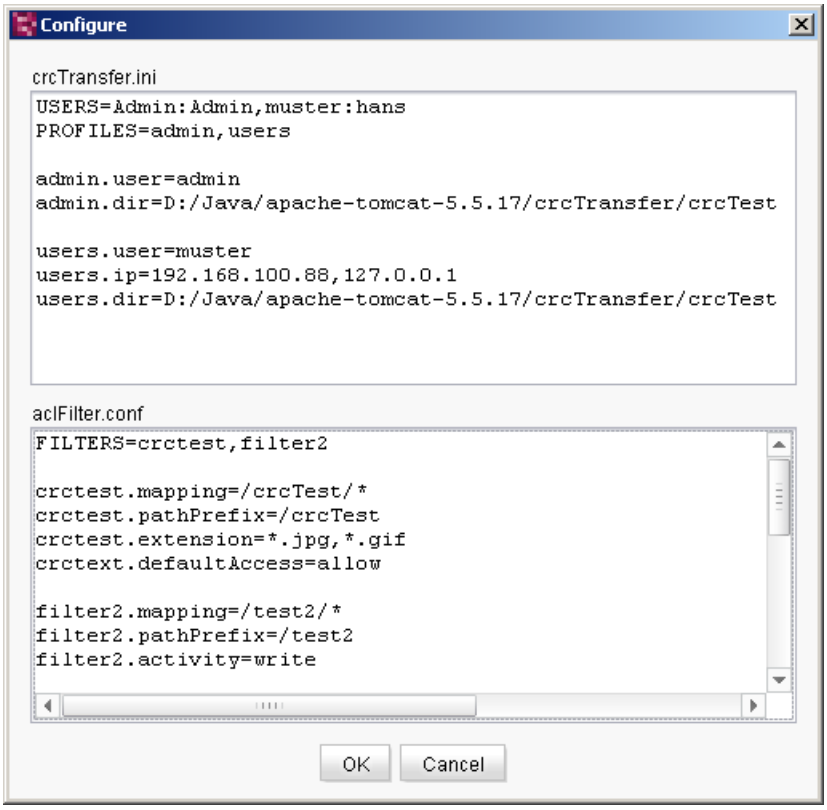

Figure 3-3: Configuring the Security web application

After changing the configuration the configuration file must be deployed on the web server. To this end the "FirstSpirit Security WebApp" web application must be updated in the web components area:

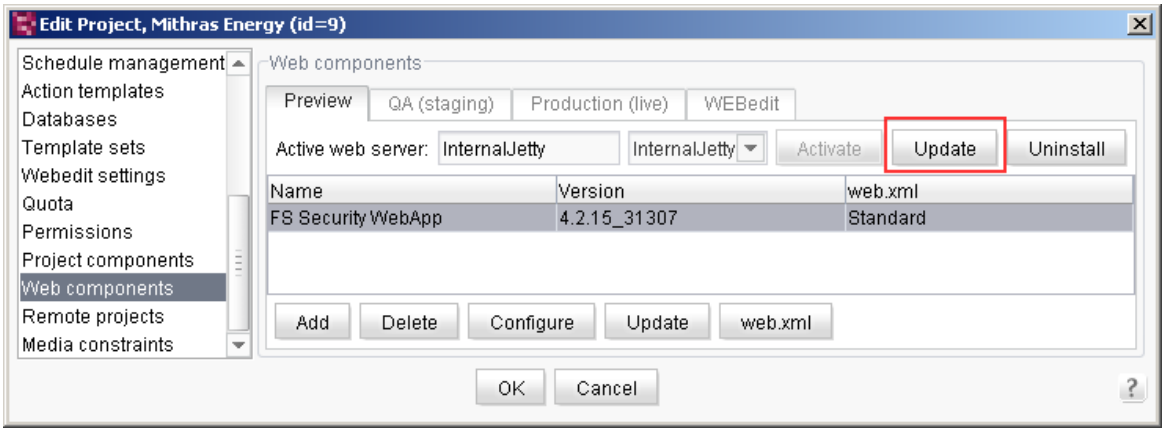

Figure 3-4: Updating the web application after configuration

3.1.4.1 crcTransfer.ini

The configuration file crcTransfer.ini is used to configure the FirstSpirit™ deployment servlet (see Chapter 3.1.3.1 page 13).

crcTransfer.ini

```
USERS=Admin:Admin,muster:hans
PROFILES=admin,users

admin.user=admin
admin.dir=D:/Java/apache-tomcat-5.5.17/crcTransfer/crcTest

users.user=muster
users.ip=192.168.100.88,127.0.0.1
users.dir=D:/Java/apache-tomcat-5.5.17/crcTransfer/crcTest
```

Figure 3-5: crcTransfer.ini configuration

There the destination directory for the deployment (DocumentRoot of the web server) as well as the login data (user name and password) can be configured.

The following parameters are available for this:

"USERS" parameter: The user accounts which can be used in the server and project configuration in the schedule for publication can be entered here. Several entries are made as a comma-separated list, whereby the user name and password of a user are each separated by a colon `:`
UserName:Password,UserName2:Password2

"PROFILES" parameter: Comma-separated profile names which are to be activated are entered here. The profiles themselves are then defined through their parameter "profilname.*".

"profilname.user" parameter: Comma-separated user names (see USERS) which are to be valid for this profile are entered here.

"profilname.ip" parameter: If comma-separated IP addresses are given here, this profile can only be used from these IP addresses.

"profilname.dir" parameter: Comma-separated absolute paths (fully qualified path) in which the servlet may be written are given here. For example, to configure deployment for the web application "fs5staging_1921116", the following path should be entered so that the `/WEB-INF` directory of the web application is not removed from the deployment schedule:

`D:/Tomcat/webapps`/fs5staging_1921116/fs_security

When entering paths in Windows file systems it is important to ensure that forward slashes are used instead of back slashes:
`D:/Tomcat/webapps`/fs5staging_1921116/fs_security

The directories given here would be completely deleted on deployment!

After the configuration of the `crcTransfer.ini` the web application should be started.

The servlet details are then located in the web.xml file of the web application:

```
<servlet-mapping>

          <servlet-name>fss-CRCTransferServlet</servlet-name>

          <url-pattern>*.CRCTransfer</url-pattern>

</servlet-mapping>
```

The servlet can be opened in the web browser to test availability of the servlet, e.g. via:
http://www.mydomain.de/ fs5staging_1921116/do.CRCTransfer
The servlet will display an error message, as no login data has been transferred into the browser.

The filter for the delivery must also be configured to ensure access protection for media within the generated project content (see Chapter 3.1.4.2 page 18).

3.1.4.2 Configuring the multi-access control filter

aclFilter.conf

```
FILTERS=fs4security_secureMedia_DE,fs4security_secureMedia_EN,fs4security_secureMedia_pic

# all language-dependent media in the folder secure media DE
fs4security_secureMedia_DE.mapping=/fs4staging_1921116/fs4_security/media/de/secure_media/*
fs4security_secureMedia_DE.pathPrefix=/fs4_security
fs4security_secureMedia_DE.defaultAccess=deny

# all language-dependent media in the folder secure media EN
fs4security_secureMedia__EN.mapping=/fs4staging_1921116/fs4_security/media/en/secure_media/*
fs4security_secureMedia__EN.pathPrefix=/fs4_security
fs4security_secureMedia__EN.defaultAccess=deny

# one language-independent medium
fs4security.mapping=/fs4staging_1921116/fs4_security/media/secure_media/picture.jpg
fs4security.pathPrefix=/fs4_security
fs4security.defaultAccess=deny
```

Figure 3-6: aclFilter.conf configuration

FILTERS: The parameter can be used to specify an identifier for the filter to be configured. The filter configuration then follows the syntax:
Identifier.ConfigurationParameter
Any number of filters can be given as a comma-separated list.

filter.mapping: Assignment of a generation directory to the filter (or more generally, details of an absolute path in the web application). The contents of the directory given here are checked by the filter. Mapping of the filter is always based on:

- Mapping must contain the servlet context of the web application (here /`fs5staging_1921116`)
- Prefix (for completing the ACL information in the live system) (here /`fs_security`)
- Path to the object to be protected (directory and/or file)

The filter chosen should always be as narrow as possible to ensure high performance during delivery into the live system.

Only one mapping may ever be defined for each filter. If several mappings are required, for example for language-dependent content, a new filter must be defined for each mapping.

If the wildcard * is given as the end of the mapping (see Figure 3-6), the whole directory including all subordinate elements is checked. If the wildcard is not given the filter affects precisely one file

only, e.g.:
`filter.mapping=/fs5staging_1921116/fs_security/media/de/secure_media/b.jpg`

The mapping must always be individually adjusted and is highly dependent on the defined path compilation in the generation (default, Infix, multiview):

- Default: In this method a separate subfolder is generated on the web server for each language of the project (de, en, etc.). For the filter mapping it must be noted that language-dependent content is generated in different folders. A separate filter with adjusted mapping must then be defined for each folder.
- Multiview and Infix: With these methods there are no language-specific subfolders, instead the files for each language are denoted by the respective language abbreviation, e.g. index.html.de, or index.de.html. Therefore, one filter is sufficient to include all the required contents – language dependent or language independent.

Examples of filter mapping (with standard "default" path generation):

The examples show the filter configuration for a specific media folder of the project. The prefix for the example configuration is `/fs_security`, the directory to be protected is `secure_media`. The corresponding folder must be given for the mapping (starting from the root node), i.e. for example:
`filter.mapping=/fs5staging_1921116/fs_security/media/de/secure_media/*`

For the filter mapping, it should be noted that language-dependent media are filed in other directories to language-independent media. This means different filter configurations are required to protect all the media in the `secure_media` directory:

Example of a language-independent filter:
Filter configuration for language-independent media in the destination folder `secure_media`:

```
FILTERS=fssecurity_1

fssecurity_1.mapping=/fs5staging_1921116/fs_security/media/secure_media/*

fssecurity_1.pathPrefix=/fs_security

fssecurity_1.defaultAccess=deny
```

Example of a language-dependent filter:
Filter configuration for language-dependent media in the languages DE and EN in the destination folder `secure_media`:

```
FILTERS=fssecurity_DE,fssecurity_EN



fssecurity_DE.mapping=/fs5staging_1921116/fs_security/media/de/secure_media/*

fssecurity_DE.pathPrefix=/fs_security

fssecurity_DE.defaultAccess=deny



fssecurity_EN.mapping=/fs5staging_1921116/fs_security/media/en/secure_media/*

fssecurity_EN.pathPrefix=/fs_security

fssecurity_EN.defaultAccess=deny
```

This means a filter entry must be made for each language.

Example of a filter for a language-independent medium:
Filter configuration for precisely one language-independent medium

```
FILTERS=fssecurity

fssecurity.mapping=/fs5staging_1921116/fs_security/media/secure_media/picture.jpg

fssecurity.pathPrefix=/fs_security

fssecurity.defaultAccess=deny
```

filter.pathPrefix: Assignment of a prefix which is to be used to supplement the path within the access control database.
Background: The database can manage several FirstSpirit™ projects. Clear, unique identification is then achieved by using the complete path including prefix. The parameter `pathPrefix` defines from which part of the filter taken into account by the filter the permission check is to take place in the access control database.

Example:
http://myServer:8000/fs5staging_1921116/fs_security/de/homepage/index.jsp
The check in the access control database then takes place from the following path:
/fs_security/de/homepage/index.jsp

The path defined here must correspond to the setting in the "ProjectProperties/Permissions" area (cf. “Access Control Database prefix" in Chapter 3.1.1).

filter.extensions: Definition of the file extensions (endings) to be checked by the filter. The file extensions can be transferred as a comma-separated list, e.g.:
`filter.extensions=*.jpg,*.png,*.gif,*.jsp`

If the parameter is NOT given, all contents of the generation directory given under `filter.mapping` are filtered.

If the filtering is limited to specific file extensions (e.g. JSP files), automatic completion of the servlet engine can occur (by specifying "Welcome Files") so that these pages are nevertheless displayed.
This means the following invocation is protected by the filter: http://myServer:8000/fs5staging_1921116/fs_security/de/homepage/index.jsp

But not the invocation:
http://myServer:8000/fs5staging_1921116/ fs_security/de/homepage/
which subsequently displays the actually protected JSP page.

This is not because of an error in FirstSpirit™, but is a configuration problem. In this case the restriction to specific file extensions should not be given.

filter.defaultAccess: This parameter can be used to define the standard (default) access behaviour of the filter if no rights or permissions are stored for the object via the permissions variable or if no entry exists in the access control database. (For example, this applies to folders as the access control database manages information about files only.)
If the value `allow` is defined, access to the filtered content is allowed for all users (only if no other permissions have been defined using the permissions variable).
If the value `deny` is defined, access is always prevented if no access rights have been defined for the object.

filter.activity: This parameter can be used to define a specific access right which is maintained using the input component for access rights (CMS_INPUT_PERMISSION). Only the access rights defined in the form area of the input component within the <ACTIVITIES> tags can be taken into account, e.g.:

```
<CMS_INPUT_PERMISSION ...>
  <ACTIVITIES>
    <ACTIVITY name="read"/>
            ...
    <ACTIVITY name="write"/>
            ...
  </ACTIVITIES>
      ...
</CMS_INPUT_PERMISSION>
```

FirstSpirit™ then decides, on the basis of the input component value in the metadata tab, whether a user has access to an object or not.

4 Classes and Methods

The following classes are available for access to the access control database:

- "AccessControlDb" class ("de.espirit.firstspirit.acl.db" package):
  The class contains methods for access to the access control database, e.g. to request one or several ACLs
  (cf. Chapter 4.1.1 page 23).
- "Acl" class ("de.espirit.firstspirit.acl" package):
  The class contains methods for obtaining the stored permission information (user permissions or access rights) on one or several files
  (cf. Chapter 4.1.2 page 37).
- "Activity" class ("de.espirit.firstspirit.acl" package):
  The class contains method for the output and/or evaluation of permissions
  (cf. Chapter 4.1.3 page 43).
- "File" class ("de.espirit.firstspirit.acl" package):
  The class contains methods for output of the stored information on a generated file, e.g. the ID, the unique identifier, etc.
  (cf. Chapter 4.1.4 page 47).

The classes which are described in this Chapter, are not in the public FirstSpirit API and will change in future versions.

4.1.1 "AccessControlDb" class ("de.espirit.firstspirit.acl.db" package)

The "AccessControlDb" class from the "de.espirit.firstspirit.acl.db" package enables access to the local access control database or the access control database contained in the web application via a script or a JSP page.

4.1.1.1 Access to the access control database (JSP page)

If the access control servlet has been registered in the web application (cf. Chapter 3.1.3.2 page 14), the servlet makes the access control database available within the application context ("application" object). The name of the attribute in the application context is defined by the ACCESS_CONTROL_DB constant of the "AccessControlServlet" class.

The invocation for loading the access control database within a JSP page is therefore:

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb"
%>
<%@
  page
  import="de.espirit.firstspirit.acl.servlet.AccessControlServlet"
%>

<%
  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);
%>
```

4.1.1.2 Access to the access control database (script)

The access control database is already available in a schedule (in the context within a script). To access the access control database, the "getAccessControlDb()" method must be opened on the "context" object. It is not necessary to give an import instruction:

```
db = context.getAccessControlDb();
```

4.1.1.3 Methods overview

The most important methods of the "AccessControlDb" class are:

- "getAcls()" method:
  Returns a list of all ACLs
  (cf. Chapter 4.1.1.4 page 25).
- "getFiles(Acl)" method:
  Returns a list of all files of an ACL
  (cf. Chapter 4.1.1.5 page 26).
- "getFile(String)" method:
  Returns a specific file on the basis of its path
  (cf. Chapter 4.1.1.6 page 27).
- "getAcl(long)" method:
  Returns a specific ACL on the basis of its ID
  (cf. Chapter 4.1.1.7 page 28).
- "deleteDir(String)" method:
  Deletes all files defined by a path
  (cf. Chapter 4.1.1.8 page 29).
- "deleteFile(File)" method:
  Deletes a file
  (cf. Chapter 4.1.1.9 page 30)
- "deleteAcl(long)" method:
  Deletes a specific ACL defined by its ID
  (cf. Chapter 4.1.1.10 page 31).
- "deleteOldFiles(String, long)" method:
  Deletes all files of a path which are older than the given date.
  (cf. Chapter 4.1.1.11 page 32).
- "getFiles(String, long)" method:
  Returns all files of a path with are younger or equal to the given date.
  (cf. Chapter 4.1.1.12 page 33).
- "getFiles(long)" method:
  Returns all files on the basis of the ID
  (cf. Chapter 4.1.1.13 page 35).

4.1.1.4 "getAcls()" method

The "getAcls()" method returns all ACLs of an access control database as a list ("List<Acl>"). Transfer parameters do not have to be specified.

Method signature:

```
public java.util.List<de.espirit.firstspirit.acl.Acl> getAcls() throws
com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getAcls():List<Acl>
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;
%>
```

4.1.1.5 "getFiles(Acl)" method

The "getFiles(Acl)" method returns a list of all files for the transferred ACL ("List<File>").

Method signature:

```
public java.util.List<de.espirit.firstspirit.acl.File> getAcls(final
de.espirit.firstspirit.acl.Acl acl) throws com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getFiles(Acl):List<File>
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl : acls) {
    files = db.getFiles(acl);
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final List<File> files = db.getFiles(acl);

    }

  }

%>
```

4.1.1.6 "getFile(String)" method

The "getFile(String)" method returns a specific file (of all ACLs). The complete path of the file must be specified as the transfer parameter.

The local access control database does not use a prefix, however the access control database in the web application does. For a deployment the prefix given in the project must therefore be placed in front of the path.

Method signature:

```
public de.espirit.firstspirit.acl.File getFile(final java.lang.String path)
throws com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getFile(String):File
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final File file =
    db
      .getFile("$CMS_VALUE(deploymentPrefix)$$CMS_REF(#global.ref, abs:2)$");
%>
```

In the example, "deploymentPrefix" is a structure variable which was defined in the Site Store. The variable is overwritten in the schedule with the corresponding prefix.

4.1.1.7 "getAcl(long)" method

The "getAcl(long)" method returns a specific ACL. The parameter to be given is the unique id of an ACL.

As assignment of the IDs for the individual ACLs is consecutive and the IDs are assigned on each generation, the method is not suitable for use over several generations. It is therefore recommended that the "getAcls()" method (cf. Chapter 4.1.1.4 page 25) or "getFile(String)" method (cf. Chapter 4.1.1.6 page 27) be used in scripts and JSP pages. An exception to this rule is determination of the ACL if the file has been obtained with "getFile(String)" and was opened with the "getAcl()" method.

Method signature:

```
public de.espirit.firstspirit.acl.Acl getAcl(final long id) throws
com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getAcl(long):Acl
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final Acl acl = db.getAcl(1L);
%>
```

4.1.1.8 "deleteDir(String)" method

The "deleteDir(String)" method can be used, e.g. after changing the prefix, to delete all files from the access control database whose path begins with the transferred path.

The local access control database does not use a prefix, however the access control database in the web application does. For a deployment the prefix given in the project must therefore be placed in front of the path.

Method signature:

```
public void deleteDir(final java.lang.String path) throws
com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
deleteDir(String):void
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  db.deleteDir("/fs_security");
%>
```

4.1.1.9 "deleteFile(File)" method

The "deleteFile(File)" method can be used to delete the given file from the access control database.

Method signature:

```
public void deleteFile(final de.espirit.firstspirit.acl.File file) throws
com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
deleteFile(File):void
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final File file =
    db
      .getFile("/fs_security/de/index.html");

  db.deleteFile(file);
%>
```

4.1.1.10 "deleteAcl(long)" method

The "deleteAcl(long)" method can be used to delete a specific ACL from the access control database defined by its ID.

Method signature:

```
public void deleteAcl(final long id) throws com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
deleteAcl(long):void
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  db.deleteAcl(1L);
%>
```

4.1.1.11 "deleteOldFiles(String, long)" method

The "deleteOldFiles(String, long)" method is used to delete all files older than the transferred date and whose path begins with the transferred path from the access control database.

The method expects the path of the files to be deleted as the first transfer parameter. The specification "null" means that all files are to be taken into account (without consideration of a prefix). The second parameter to be given is the date as a time stamp (data type: "long"). A readable date format can be transferred into a time stamp using Java methods.

For removal of the files, the transferred time stamp is compared with the time stamp of the last update of the stored data of an object (cf. Chapter 4.1.4.7 page 56). If the time stamp of the last update of the stored data is smaller than the transferred time stamp and if the object's path begins with the transferred path the object or file is deleted.

The local access control database does not use a prefix, however the access control database in the web application does. For a deployment the prefix given in the project must therefore be placed in front of the path.

Method signature:

```
public void deleteOldFiles(final java.lang.String pathPrefix, final long
timestamp) throws com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
deleteOldFiles(String, long):void
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  db.deleteOldFiles("/fs_security", 1196674444102L);
%>
```

4.1.1.12 "getFiles(String, long)" method

The "getFiles(String, long)" method returns a list of all files whose paths begin with the transferred path and which are just as old or are younger than the transferred date.

The method expects the path of the files as the first transfer parameter. The specification "null" means that all files are to be taken into account (without consideration of a prefix). The second parameter to be given is the date as a time stamp (data type: "long"). A readable date format can be transferred into a time stamp using Java methods.

For determination of the files, the transferred time stamp is compared with the time stamp of the last update of the stored data of an object (cf. Chapter 4.1.4.7 page 56). If the time stamp of the last update of the stored data is smaller than the given time stamp and if the object's path begins with the transferred path the object or file is returned.

The local access control database does not use a prefix, however the access control database in the web application does. For a deployment the prefix given in the project must therefore be placed in front of the path.

The "getFiles(String, long)" method can be used in a script of a schedule to determine data of all generated files. The starting time of the generation is to be used for this. The starting time of the generation is availalbe in the script context of a schedule with the help of the "getStartTime()" method. The "getStartTime()" method returns a "java.util.Date" object. An example of use of "getStartTime()" is given below.

Method signature:

```
public java.util.List<de.espirit.firstspirit.acl.File> getFiles(final
java.lang.String pathPrefix, final long timestamp) throws
com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getFiles(String, long):List<File>
```

Example (script):

```
import java.text.SimpleDateFormat;
import java.util.Locale;

db = context.getAccessControlDb();
files =
  db != null
  ? db.getFiles(null, context.getStartTime().getTime())
  : null;

if (files != null) {
  df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);
  for (file : files) {
    print(
         "   path=" + file.getPath()
      + ", lastModified=" + df.format(file.getLastModified())
      + ", lastUpdate=" + df.format(file.getLastUpdate())
      + ", crc=" + file.getCrc32()
    );
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);


  final List<File> files =
    db
      .getFiles("/fs_security", 1196675825992L);
%>
```

4.1.1.13 "getFiles(long)" method

The "getFiles(long)" method returns a list of all files corresponding to the transferred ID.

Several files or objects have the same ID if several files are generated by the generation (starting from an element). This is the case, e.g. for a data source section. Under certain circumstances, several files are generated for the data records on generation of the page reference in the Site Store. The ID of the files is therefore the same (the prefix and creation date of the files can however be different). In this case the ID of the page reference must be given to obtain a list of all files.

Method signature:

```
public java.util.List< de.espirit.firstspirit.acl.File File> getFiles(final
long elementId) throws com.sleepycat.je.DatabaseException
```

Method signature (abbreviated form):

```
getFiles(long):List<File>
```

Example (script):

```
import java.text.SimpleDateFormat;
import java.util.Locale;

db = context.getAccessControlDb();
files =
  db != null
  ? db.getFiles(123456L)
  : null;

if (files != null) {
  df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);
  for (file : files) {
    print(
         "   path=" + file.getPath()
      + ", lastModified=" + df.format(file.getLastModified())
      + ", lastUpdate=" + df.format(file.getLastUpdate())
      + ", crc=" + file.getCrc32()
    );
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);


  final List<File> files =
    db
      .getFiles(123456L);
%>
```

4.1.2 "Acl" class ("de.espirit.firstspirit.acl" package)

The "Acl" class from the "de.espirit.firstspirit.acl" package enables access to the stored permission information (user permissions and/or access rights) on one or several files.

Font images do not contain any permission information.

4.1.2.1 Methods overview

The most important methods of the "Acl" class are:

- "getId()" method:
  Returns the ID of an ACL
  (cf. Chapter 4.1.2.2 page 38).
- "getDocument()" method:
  Returns the symbolic name of the group hierarchy used (e.g. in the rights permission input component)
  (cf. Chapter 4.1.2.3 page 39).
- "getActivities()" method:
  Returns the configured and stored activities
  (cf. Chapter 4.1.2.4 page 40).
- "getPermissions()" method:
  Returns the stored permission definition for the individual activities
  (cf. Chapter 4.1.2.5 page 41).
- "getPriority()" method:
  Returns the configured and stored standard if rights (permissions) overlap.
  (cf. Chapter 4.1.2.6 page 42).

4.1.2.2 "getId()" method

The "getId()" method returns the ID of an ACL.

Method signature:

```
public long getId()
```

Method signature (abbreviated form):

```
getId():long
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl " %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
%><%= acl.getId() %><br /><%
    }
  }
%>
```

4.1.2.3 "getDocument()" method

The "getDocument()" method returns the symbolic name of the group hierarchy used. The symbolic name of the group hierarchy used is defined, among other things, in the permission input component with the "group" parameter:

```
<CMS_INPUT_PERMISSION name="st_permission" group="GruppenFile">
  <ACTIVITIES>
    <ACTIVITY name="read"/>
    <ACTIVITY name="write"/>
  </ACTIVITIES>
  <LANGINFOS>
    <LANGINFO label="CMS_INPUT_PERMISSION" lang="*"/>
  </LANGINFOS>
</CMS_INPUT_PERMISSION>
```

Method signature:

```
public java.lang.String getDocument()
```

Method signature (abbreviated form):

```
getDocument():String
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
%><%= acl.getDocument() %><br /><%
    }
  }
%>
```

4.1.2.4 "getActivities()" method

The "getActivities()" method returns the defined and stored activity identifier (part of the access rights) as a list. Activities are defined, among other things, in the permission input component with the "<ACTIVITIES>" tag:

```
<CMS_INPUT_PERMISSION name="st_permission" group="GruppenFile">
  <ACTIVITIES>
    <ACTIVITY name="read"/>
    <ACTIVITY name="write"/>
  </ACTIVITIES>
  <LANGINFOS>
    <LANGINFO label="CMS_INPUT_PERMISSION" lang="*"/>
  </LANGINFOS>
</CMS_INPUT_PERMISSION>
```

Each activity can have different permissions.

Method signature:

```
public List<java.lang.String> getActivities()
```

Method signature (abbreviated form):

```
getActivities():List<String>
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {
    for (final Acl acl: acls) {
%><%= acl.getActivities() %><br /><%
    }
  }
%>
```

4.1.2.5 "getPermissions()" method

The "getPermissions()" method returns a map. The map contains all permission information on the corresponding files. The key of a map entry is an activity identifier (cf. also Chapter 4.1.2.4 page 40) and the value is a permission object ("Activity"). The permission object can then be used to evaluate the permissions of an activity.

Method signature:

```
public java.util.Map<java.lang.String, de.espirit.firstspirit.acl.Activity>
getPermissions()
```

Method signature (abbreviated form):

```
getPermissions():Map<String, Activity>
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
%><%= acl.getPermissions() %><br /><%
    }
  }
%>
```

4.1.2.6 "getPriority()" method

The "getPriority()" method returns the defined and stored value for the conflict response if permission configurations for a permission object overlap. The default value for the conflict response is defined, among other things, in the permission input parameter with the "priority" parameter:

```
<CMS_INPUT_PERMISSION name="st_permission" priority="deny"
group="GruppenFile">
  <ACTIVITIES>
    <ACTIVITY name="read"/>
    <ACTIVITY name="write"/>
  </ACTIVITIES>
  <LANGINFOS>
    <LANGINFO label="CMS_INPUT_PERMISSION" lang="*"/>
  </LANGINFOS>
</CMS_INPUT_PERMISSION>
```

Method signature:

```
public de.espirit.firstspirit.service.permission.Priority getPriority()
```

Method signature (abbreviated form):

```
getPriority():Priority
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
%><%= acl.getPriority() %><br /><%
    }
  }
%>
```

4.1.3 "Activity" class ("de.espirit.firstspirit.acl" package)

The "Activity" class from the "de.espirit.firstspirit.acl" package can be used to evaluate and further process stored activity permission information of one or several files.

Activities are defined, among other things, in the permission input component. The authorisations or user permissions are then defined with the permission input component on the respective objects in the individual stores (meta information).

4.1.3.1 Methods overview

The most important methods of the "Activity" class are:

- "getName()" method:
  Returns the stored activity identifier
  (cf. Chapter 4.1.3.2 page 44).
- "getAllowed()" method:
  Returns the users/groups for whom access to the files is allowed
  (cf. Chapter 4.1.3.3 page 45).
- "getForbidden()" method:
  Returns the users/groups for whom access to the files is forbidden
  (cf. Chapter 4.1.3.4 page 46).

4.1.3.2 "getName()" method

The "getName()" method returns the names of the activity.

Method signature:

```
public java.lang.String getName()
```

Method signature (abbreviated form):

```
getName():String
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.Activity,
    java.util.Map" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final Map<String, Activity> permissions =
        acl.getPermissions();

        if (permissions != null) {
          for(final Activity activity : permissions.values()) {
%><%= activity.getName() %><br /><%
          }
        }

    }
  }
%>
```

4.1.3.3 "getAllowed()" method

The "getAllowed()" method is used to return the groups and/or users for whom access to one or several files is allowed as a list.

Method signature:

```
public java.util.List<java.lang.String> getAllowed()
```

Method signature (abbreviated form):

```
getAllowed():List<String>
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.Activity,
    java.util.Map" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final Map<String, Activity> permissions =
        acl.getPermissions();

        if (permissions != null) {
          for(final Activity activity : permissions.values()) {
%><%= activity.getAllowed() %><br /><%
          }
        }

    }
  }
%>
```

4.1.3.4 "getForbidden()" method

The "getForbidden()" method is used to return the groups and/or users for whom access to one or several files is forbidden as a list.

Method signature:

```
public java.util.List<java.lang.String> getForbidden()
```

Method signature (abbreviated form):

```
getForbidden():List<String>
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.Activity,
    java.util.Map" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final Map<String, Activity> permissions =
        acl.getPermissions();

        if (permissions != null) {
          for(final Activity activity : permissions.values()) {
%><%= activity.getForbidden() %><br /><%
          }
        }

    }
  }


%>
```

4.1.4 "File" class ("de.espirit.firstspirit.acl" package)

The "file" class from the "de.espirit.firstspirit.acl" package makes it possible to access the information of a generated file. The following information is available:

- the complete path to the object
- the object's CRC-32 checksum
- the date of the last change to the object
- the date of the last update of an object's stored data
- the object's unique identifier
- the object's ID

4.1.4.1 Methods overview

The most important methods of the "File" class are:

- "getPath()" method:
  Returns the complete path to the object (cf. Chapter 4.1.4.2 page 48).
- "getElementId()" method:
  Returns the object's ID (cf. Chapter 4.1.4.3 page 49).
- "getElementUid()" method:
  Returns the object's unique identifier
  (cf. Chapter 4.1.4.4 page 51).
- "getCrc32()" method:
  Returns the object's CRC-32 checksum (cf. Chapter 4.1.4.5 page 53).
- "getLastModified()" method:
  Returns the date of the last change to the object (cf. Chapter 4.1.4.6 page 54).
- "getLastUpdate()" method:
  Returns the date of the last update of an object's stored data (cf. Chapter 4.1.4.7 page 56).
- "getAcl()" method:
  Returns the ACL's ID (cf. Chapter 4.1.4.8 page 58).
- "getLength()" method:
  Returns the file size (cf. Chapter 4.1.4.9 page 60).
- "getElementTag()" method:
  Returns the tag name of an object (cf. Chapter 4.1.4.10 page 61).

4.1.4.2 "getPath()" method

The "getPath()" method is used to return the stored, absolute path up to the generation root node of an object.

The local access control database does not use a prefix, however the access control database in the web application does. On deployment the prefix given in the project is therefore placed in front of the path and is taken into account on saving.

Method signature:

```
public String getPath()
```

Method signature (abbreviated form):

```
getPath():String
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getPath());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getPath() %><br /><%
        }
      }

    }
  }


%>
```

4.1.4.3 "getElementId()" method

The "getElementId()" method is used to return the ID which was valid on generation of the object.

Font images do not have an ID. In this case "-1" is returned.

Method signature:

```
public long getElementId()
```

Method signature (abbreviated form):

```
getElementId():long
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getElementId());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getElementId() %><br /><%
        }
      }

    }
  }
%>
```

4.1.4.4 "getElementUid()" method

The "getElementUid()" method is used to return the unique identifier which was valid on generation of the object.

Unique identifiers are object-dependent and are not unique throughout the project, i.e. for example media and page references can have the same unique designator.

Font images do not have a unique identifier. In this case "null" is returned.

Method signature:

```
public java.lang.String getElementUid()
```

Method signature (abbreviated form):

```
getElementUid():String
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getElementUid());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getElementUid() %><br /><%
        }
      }

    }
  }


%>
```

4.1.4.5 "getCrc32()" method

The "getCrc32()" method returns the calculated CRC-32 checksum of a generated file.

CRC-32 checksums are determined on the basis of the file content.

Method signature:

```
public long getCrc32()
```

Method signature (abbreviated form):

```
getCrc32():long
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getCrc32());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%


  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;
```

```
  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getCrc32() %><br /><%
        }
      }

    }
  }

%>
```

4.1.4.6 "getLastModified()" method

The "getLastModified()" method returns the date and/or time of the last change to the object (as a time stamp). It is returned in "long" form and can be transferred by Java methods into a readable date format.

Method signature:

```
public long getLastModified()
```

Method signature (abbreviated form):

```
getLastModified():long
```

Example (script):

```
import java.text.SimpleDateFormat;
import java.util.Locale;

df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(
          file.getLastModified()
          + " ("
          + df.format(file.getLastModified())
          + ")"
        );
      }
    }
```

```
  }
}
```

Example (JSP page):

```
<%@
  page
  import="java.text.SimpleDateFormat,
    java.util.Locale,
    de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final SimpleDateFormat df =
    new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%>
  <%= file.getLastModified() %>
  (<%= df.format(file.getLastModified()) %>)
  <br />
<%
        }
      }

    }
  }


%>
```

4.1.4.7 "getLastUpdate()" method

The "getLastUpdate()" method returns the date and/or time of the last update of the object's stored data (as a time stamp). It is returned in "long" form and can be transferred by Java methods into a readable date format.

Method signature:

```
public long getLastUpdate()
```

Method signature (abbreviated form):

```
getLastUpdate():long
```

Example (script):

```
import java.text.SimpleDateFormat;
import java.util.Locale;

df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);
db = context.getAccessControlDb();

db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(
          file.getLastUpdate()
          + " ("
          + df.format(file.getLastUpdate())
          + ")"
        );
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="java.text.SimpleDateFormat,
    java.util.Locale,
    de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final SimpleDateFormat df =
    new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.GERMANY);

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%>
  <%= file.getLastUpdate() %>
  (<%= df.format(file.getLastUpdate()) %>)
  <br />
<%
        }
      }
    }
  }

%>
```

4.1.4.8 "getAcl()" method

The "getAcl()" method can be used to determine the ID of the ACL to which the file or the object belongs.

The ID can then be used to determine the file's permission information.

Method signature:

```
public long getAcl()
```

Method signature (abbreviated form):

```
getAcl():long
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(
          file.getPath()
          + ": "
          + db.getAcl(file.getAcl()).getActivities()
        );
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import=" de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {

      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%>
  <%= file.getPath() %>:
  <%= db.getAcl(file.getAcl()).getActivities() %>
  <br />
<%
        }
      }
    }
  }
%>
```

4.1.4.9 "getLength()" method

The "getLength()" method returns the size of a file in bytes.

Method signature:

```
public long getLength()
```

Method signature (abbreviated form):

```
getLength():long
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getLength());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getLength() %><br /><%
        }
      }
    }
```

```
  }
%>
```

4.1.4.10 "getElementTag()" method

The "getElementTag()" method can be used to determine the tag name of a file or of an object. This method is important to be able to differentiate between objects as, e.g. the UID of a page reference and of a medium can be identical.

Method signature:

```
public String getElementTag()
```

Method signature (abbreviated form):

```
getElementTag():String
```

Example (script):

```
db = context.getAccessControlDb();
acls = db != null ? db.getAcls() : null;

if (acls != null) {
  for (acl: acls) {
    files = db.getFiles(acl);
    if (files != null) {
      for (file : files) {
        print(file.getElementTag());
      }
    }
  }
}
```

Example (JSP page):

```
<%@
  page
  import="de.espirit.firstspirit.acl.db.AccessControlDb,
    de.espirit.firstspirit.acl.servlet.AccessControlServlet,
    java.util.List,
    de.espirit.firstspirit.acl.Acl,
    de.espirit.firstspirit.acl.File" %><%

  final AccessControlDb db =
    (AccessControlDb) application
      .getAttribute(AccessControlServlet.ACCESS_CONTROL_DB);

  final List<Acl> acls = db != null ? db.getAcls() : null;

  if (acls != null) {

    for (final Acl acl: acls) {
      final List<File> files = db.getFiles(acl);
      if (files != null) {
        for (File file : files) {
%><%= file.getElementTag() %><br /><%
        }
      }
    }
  }
%>
```

5 CRC Transfer Servlet

5.1 Registering and deregistering listeners

The CRC transfer servlet offers two public methods for registering and deregistering listeners.

```
public void addTransferListener(final TransferListener
transferListener);
public void removeTransferListener(final TransferListener
transferListener);
```

The CRC transfer servlet registers itself in the ServletContext and can be reached as follows:

```
final CRCTransferServlet crcServlet = (CRCTransferServlet)
servletContext.getAttribute(CRCTransferServlet.CRC_TRANSFER_SERVLET);
```

A TransferListener is informed before and after each file operation performed by the CrcServlet. At the same time the listener invocations block the current servlet invocation. Prolonged operations should therefore be executed concurrently in a thread, as a timeout occurs on the client side if the request is blocked for too long.

The classes and interfaces which are of interest for implementation of a listener are introduced in the following chapter (see Chapter 5.2 page 63).

5.2 Implementation of a listener

The following classes/interfaces are available for the implementation of listeners:

- "de.espirit.firstspirit.client.io.crc.TransferListener" interface:
- "de.espirit.firstspirit.client.io.crc.AbstractTransferListener" class
- "de.espirit.firstspirit.client.io.crc.CRCSession" interface

5.3 Example

The example JSP page must run in the same web application as the CrcServlet and implements a simple listener which outputs all calls on system.out:

Parameters of the JSP page:

- TransferListenerTest.jsp: Output CrcServlet and listeners
- TransferListenerTest.jsp?action=add Register listeners
- TransferListenerTest.jsp?action=remove deregister listeners.

```
<%@ page import="de.espirit.firstspirit.client.io.crc.*, java.util.Date"

%><pre><%

final CRCTransferServlet crcServlet = (CRCTransferServlet)
pageContext.getServletContext().getAttribute(CRCTransferServlet.CRC_TRANSFER_S
ERVLET);
      final String action = request.getParameter("action");
      if ("add".equals(action)) {
            final TransferListener listener = new AbstractTransferListener() {

public void beforeListFiles(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeListFiles");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                  }

public void beforeGetFile(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeGetFile");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                  }

public void beforeUpload(final CRCSession session, final String path, final
boolean append) {
                        System.out.println("JspClass.beforeUpload");
                        System.out.println("  path=" + path);
                        System.out.println("  append=" + append);
                        dumpCRCSession(session);
                  }

public void beforeDownload(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeDownload");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                  }

public void beforeDelete(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeDelete");
```

```
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void beforeExists(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeExists");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void beforeRename(final CRCSession session, final String oldPath, final
String newPath) {
                        System.out.println("JspClass.beforeRename");
                        System.out.println("  oldPath=" + oldPath);
                        System.out.println("  newPath=" + newPath);
                        dumpCRCSession(session);
                }

public void beforeTouch(final CRCSession session, final String path, final
long time) {
                        System.out.println("JspClass.beforeTouch");
                        System.out.println("  path=" + path);
                        System.out.println("  time=" + time);
                        dumpCRCSession(session);
                }

public void beforeSwitch(final CRCSession session, final String oldPath, final
String newPath) {
                        System.out.println("JspClass.beforeSwitch");
                        System.out.println("  oldPath=" + oldPath);
                        System.out.println("  newPath=" + newPath);
                        dumpCRCSession(session);
                }

public void beforeMkdir(final CRCSession session, final String path) {
                        System.out.println("JspClass.beforeMkdir");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterListFiles(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterListFiles");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterGetFile(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterGetFile");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterUpload(final CRCSession session, final String path, final
boolean append) {
                        System.out.println("JspClass.afterUpload");
```

```
                        System.out.println("  path=" + path);
                        System.out.println("  append=" + append);
                        dumpCRCSession(session);
                }

public void afterDownload(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterDownload");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterDelete(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterDelete");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterExists(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterExists");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

public void afterRename(final CRCSession session, final String oldPath, final
String newPath) {
                        System.out.println("JspClass.afterRename");
                        System.out.println("  oldPath=" + oldPath);
                        System.out.println("  newPath=" + newPath);
                        dumpCRCSession(session);
                }

public void afterTouch(final CRCSession session, final String path, final long
time) {
                        System.out.println("JspClass.afterTouch");
                        System.out.println("  path=" + path);
                        System.out.println("  time=" + time);
                        dumpCRCSession(session);
                }

public void afterSwitch(final CRCSession session, final String oldPath, final
String newPath) {
                        System.out.println("JspClass.afterSwitch");
                        System.out.println("  oldPath=" + oldPath);
                        System.out.println("  newPath=" + newPath);
                        dumpCRCSession(session);
                }

public void afterMkdir(final CRCSession session, final String path) {
                        System.out.println("JspClass.afterMkdir");
                        System.out.println("  path=" + path);
                        dumpCRCSession(session);
                }

private void dumpCRCSession(final CRCSession session) {
System.out.println("  user=" + session.getUser());
```

```
System.out.println("  remoteAddr=" + session.getLastRemoteAddr());
System.out.println("  creationTime=" + new Date(session.getCreationTime()));
System.out.println("  lastUsageTime=" + new Date(session.getLastUsageTime()));
                  }
      };
crcServlet.addTransferListener(listener);
final TransferListener oldListener = (TransferListener)
session.getAttribute("TransferListener");

if (oldListener != null) {
      crcServlet.removeTransferListener(oldListener);
      }
      session.setAttribute("TransferListener", listener);
      } else if ("remove".equals(action)) {
      final TransferListener listener = (TransferListener)
      session.getAttribute("TransferListener");

if (listener != null) {
      crcServlet.removeTransferListener(listener);
      session.removeAttribute("TransferListener");
      }
}

out.println("CRCTransferServlet: " + crcServlet);
out.println("TransferListener: " + session.getAttribute("TransferListener"));
%></pre>
```

6 Legal notices

The module "FirstSpirit™ Security" is a product of the e-Spirit AG, Dortmund, Germany.

When using this module only the licence agreed between the e-Spirit AG and the user is valid.

You can find information about third-party software which is potentially used for the module but not produced by the e-Spirit AG, their own licences and – as the case may be – information about updates on the start page of each FirstSpirit™ server in the area "Legal notices".