Daisy documentation

September 24, 2007

daisy

Table of Contents

1 Documentation Home

These pages contain the documentation of the Daisy 2.1 release.

See also:

- main Daisy site[1]
- documentation of other releases (page 0)

The documentation is also available published as a Daisy-book[2].

For an end-user introduction to Daisy, have a look at the video tutorials[3].

Notes

1. http://cocoondev.org/daisy/
2. /books/
3. http://svn.cocoondev.org/dist/daisy/movies/

2 Installation

2.1 Downloading Daisy

Packaged versions of Daisy can be found in the distribution area[1] (Sourceforge). This includes everything required to run Daisy, except for:

- a Java Virtual Machine (JVM): Java 1.5 or higher required
- a MySQL database: version 4.1.7 or higher required (5 also fine)

If you don't have these already, the installation of these will be covered further on.

Consider subscribing to the Daisy mailing list[2] to ask questions and talk with fellow Daisy users and developers.

There is also information available about the source code (page 43).

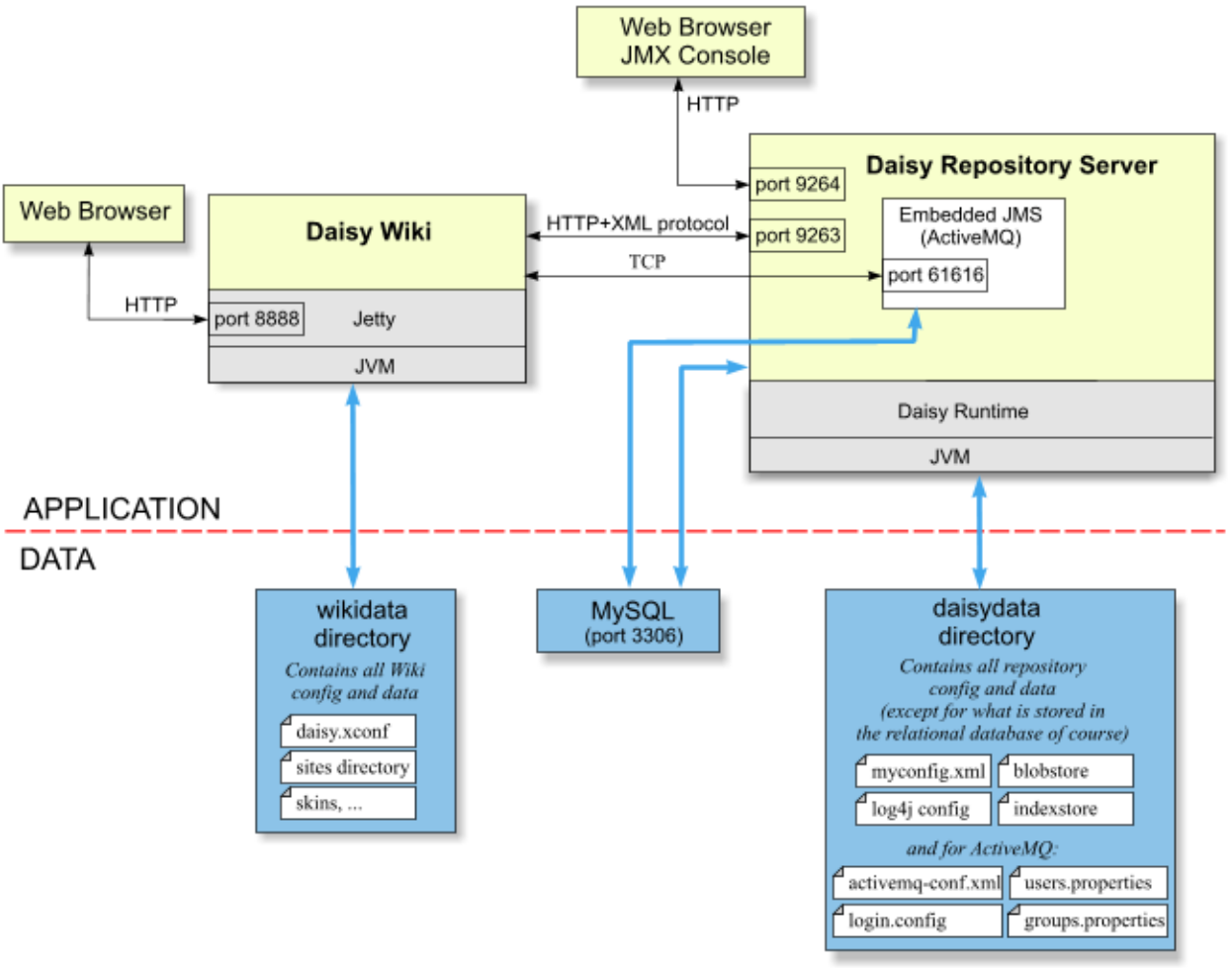
2.2 Installation Overview

Daisy is a multi-tier application, consisting of a repository server and a publication layer. Next to those, a database server (MySQL) is required. All together, this means three processes, which can run on the same server or on different servers.

The Daisy binary distribution packs most of the needed software together, the only additional things you'll need is a Java Virtual Machine for your platform, and MySQL. All libraries and applications shipped with Daisy are the original, unmodified distributions that will be configured as part of the installation. We've only grouped them in one download for your convenience.

If you follow the instructions in this document, you can have Daisy up and running in less than an hour.

The diagram below gives an overview of the the setup. All shown port numbers are configurable of course.



2.2.1 Platform Requirements

We have tested the Daisy installation on Windows 2000/XP, GNU/Linux and MacOSX. Other unixes like Solaris should also work, though we don't test that ourselves.

2.2.2 Memory Requirements

By default, the Daisy Wiki and Daisy Repository Server are started with a maximum heap size of 128 MB each. To this you need to add some overhead of the JVMs themselves, and then some memory for MySQL, the OS and its (filesystem) caches. This doesn't mean all this memory will be used, that will depend on usage intensity.

2.2.3 Required knowledge

These installation instructions assume you're comfortable with installing software, editing configuration (XML) files, running applications from the command line, setting environment variables, and that sort of stuff.

2.2.4 Can I use Oracle, PostgreSQL, MS-SQL, ... instead of MySQL? Websphere, Weblogic, Tomcat, ... instead of Jetty?

Daisy contains the necessary abstractions to support different database engines, though we currently only support MySQL. Users are welcome to contribute and maintain different databases (ask on the mailing list how to get started).

The Daisy Wiki webapp should be able to run in any servlet container (at least one that can run unpacked webapps, and as far as there aren't any Cocoon-specific issues), but we ship Jetty by default. For example, using Tomcat instead of Jetty is very simple and is described on this page (page 307).

2.3 Installing a Java Virtual Machine

Daisy requires the Java JDK or JRE 1.5 or 1.6 (the versions are also know as 5 or 6). You can download it from here on the Sun site[3] (take by preference the JDK, not the JRE). Install it now if you don't have it already.

After installation, make sure the `JAVA_HOME` environment variable is defined and points to the correct location (i.e., the directory where Java is installed). To verify this, open a command prompt or shell and enter:

```
For Windows:
%JAVA_HOME%/bin/java -version

For Linux:
$JAVA_HOME/bin/java -version
```

This should print out something like:

```
java version "1.5.0"

or

java version "1.6.0"
```

2.3.1 Installing JAI (Java Advanced Imaging) -- optional

If you want images (especially PNG) to appear in PDFs, it is highly advisable to install JAI, which you can download from the JAI project on java.net[4]. Take the JDK (or JRE) package, this will make JAI support globally available.

2.4 Installing MySQL

Daisy requires one of the following MySQL versions:

- version 4.1.7 or a newer version from the 4.1.x series
- version 5
- **what won't work**: version 3 (doesn't support transactions), version 4.0 (doesn't support subselects)

MySQL can be downloaded from mysql.com[5]. Install it now, and start it (often done automatically by the install).

Windows users can take the "Windows Essentials" package. During installation and the configuration wizard, you can leave most things to their defaults. In particular, be sure to leave the "Database Usage" to "Multifunctional Database", and leave the TCP/IP Networking enabled (on port 3306). When it asks for the default character set, select "Best Support For Multilingualism" (this will use UTF-8). When it asks for Windows options, check the option "Include Bin Directory In Windows Path".

Linux users: install the "MySQL server" and "MySQL client" packages. Installing the MySQL server RPM will automatically initialize and start the MySQL server.

2.4.1 Creating MySQL databases and users

MySQL is used by both the Daisy Repository Server and JMS (ActiveMQ). Therefore, we are now going to create two databases and two users.

Open a command prompt, and start the MySQL client as root user:

```
mysql -uroot -pYourRootPassword
```

On some systems, the root user has no password, in which case you can drop the -p parameter.

Now create the necessary databases, users and access rights by entering (or copy-paste) the commands below in the mysql client. What follows behind the `IDENTIFIED BY` is the password for the user, which you can change if you wish. The daisy@localhost entries are necessary because otherwise the default access rights for anonymous users @localhost will take precedence. If you'll run MySQL on the same machine as the Daisy Repository Server, you only need the @localhost entries.

```
CREATE DATABASE daisyrepository CHARACTER SET 'utf8';
GRANT ALL ON daisyrepository.* TO daisy@'%' IDENTIFIED BY 'daisy';
GRANT ALL ON daisyrepository.* TO daisy@localhost IDENTIFIED BY 'daisy';
CREATE DATABASE activemq CHARACTER SET 'utf8';
GRANT ALL ON activemq.* TO activemq@'%' IDENTIFIED BY 'activemq';
GRANT ALL ON activemq.* TO activemq@localhost IDENTIFIED BY 'activemq';
```

2.5 Extract the Daisy download

Extract the Daisy download. On Linux/Unix you can extract the `.tar.gz` file as follows:

```
tar xvzf daisy-<version>.tar.gz
```

On non-Linux unixes (Solaris notably), use the GNU tar version if you experience problems extracting.

On Windows, use the `.zip` download, which you can extract using a tool like WinZip.

After extraction, you will get a directory called `daisy-<version>`. This directory is what we will call from now on the **DAISY_HOME** directory. You may set a global environment variable pointing to that location, or you can do it each time in the command prompt when needed.

2.6 Daisy Repository Server

2.6.1 Initialising and configuring the Daisy Repository

Open a command prompt or shell and set an environment variable `DAISY_HOME`, pointing to the directory where Daisy is installed.

```
Windows:
set DAISY_HOME=c:\daisy-2.1

Linux:
export DAISY_HOME=/home/daisy_user/daisy-2.1
```

Then go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-repository-init
```

Follow the instructions on screen. The installation will (1) initialize the database tables for the repository server and (2) create a Daisy data directory containing customized configuration files.

2.6.2 Starting the Daisy Repository Server

Still in the same command prompt (or in a new one, but make sure `DAISY_HOME` is set), go to the directory `<DAISY_HOME>/repository-server/bin`, and execute:

```
daisy-repository-server <location-of-daisy-data-dir>
```

In which you replace `<location-of-daisy-data-dir>` with the location of the daisy data directory created in the previous step.

Starting the repository server usually only takes a few seconds, however the first time it will take a bit longer because the workflow database tables are created during startup. When the server finished starting it will print a line like this:

```
Daisy repository server started [timestamp]
```

Wait for this line to appear (the prompt will not return).

2.7 Daisy Wiki

2.7.1 Initializing the Daisy Wiki

Before you can run the Daisy Wiki, the repository needs to be initialised with some document types, a "guest" user, a default ACL configuration, etc.

Open a command prompt or shell, make sure `DAISY_HOME` is set, go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-wiki-init
```

The program will start by asking a login and password, enter here the user created during the execution of daisy-repository-init (the default was testuser/testuser). It will also ask for the URL where the repository is listening, you can simply press enter here.

If everything goes according to plan, the program will now print out some informational messages and end with "Finished.".

2.7.2 Creating a "wikidata" directory

Similar to the data directory of the Daisy repository server, the Daisy Wiki also has its own data directory (which we call the "wikidata directory").

To set up this directory, open a command prompt or shell, make sure `DAISY_HOME` is set, go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-wikidata-init
```

and follow the instructions on-screen.

Since the Daisy Wiki and the Daisy repository server are two separate applications (which might be deployed on different servers), each has its own data directory.

2.7.3 Creating a Daisy Wiki Site

The Daisy Wiki has the concept of multiple sites, these are multiple views on top of the same repository. You need at least one site to do something useful with the Daisy Wiki, so we are now going to create one.

Open a command prompt or shell, make sure `DAISY_HOME` is set, go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-wiki-add-site <location of wikidata directory>
```

The application starts by asking the same parameters as for daisy-wiki-init.

Then it will ask a name for the site. This should be a name without spaces. If you're inspirationless, enter something like "test" or "main".

Then it will ask for the sites directory location, for which the presented default should be OK, so just press enter.

2.7.4 Starting the Daisy Wiki

Open a command prompt or shell and make sure `DAISY_HOME` is set.

Go to the directory `<DAISY_HOME>/daisywiki/bin`, and execute:

```
daisy-wiki <location of wikidata directory>
```

Background info: this will start Jetty (a servlet container) with the webapp found in `<DAISY_HOME>/daisywiki/webapp`.

2.8 Finished!

Now you can point your web browser to:

```
http://localhost:8888/
```

To be able to create or edit documents, you will have to change the login, you can use the user you created for yourself while running daisy-repository-init (the default was testuser/testuser).

To start the Daisy repository server and Daisy Wiki after the initial installation, see the summary here (page 300), or even better, set up service (init) scripts (page 301) to easily/automatically start and stop Daisy.

2.9 2.0(.x) to 2.1 changes

- Replaced Avalon Merlin by something new called the "Daisy Runtime"
    - most users won't notice this
    - the Daisy Runtime is basically some thin infrastructure around Spring. See docs (page 141).
    - writing repository plugins (authentication schemes, extensions, ...) is now much better supported. Plugins can simply be dropped in the datadir/plugins directory. See docs (page 134) and tutorial for creating an authentication scheme (page 137).
    - logging is now performed using log4j instead of Avalon logkit, as a consequence the logging configuration changed.
- Source build: various simplifications:
    - no Merlin installation needed anymore because of the introduction of the new Daisy Runtime
    - made the development setup similar to the binary setup by also using the repository data directory concept
    - moved to Maven 1.1. The plugins don't need to be installed manually anymore
    - no Xalan jar in Maven install needed anymore
    - for the Wiki, added a "maven cocoon.download" goal to automate this step
    - and more
- Query language:
    - For multivalue and/or hierarchical field identifiers, an index-notation is now supported to address specific values. For example `$MyField[2][3]`. The index is 1-based. A negative index counts from the end. To specify a hierarchy index without multivalue index, the notation `$MyField[*][3]` is used.
    - New function GetLinkPath(linkFieldName, includeCurrent, linkExpr): returns a HierarchyPath obtained by recursively following a link field. Only works in evaluate mode.

    - New function ReversePath(arg): takes a HierarchyPath value as argument and returns a HierarchyPath with the same elements but in reversed order.
    - New function String(arg): evaluates its argument and converts it to a string representation.
    - Concat function: in evaluate mode, non-string arguments are now accepted which will be automatically converted to strings (with same algorithm as the String function).
    - matchesPath: the path specification argument is now evaluated dynamically, allowing to use expressions in there. For example, `matchesPath(Concat(ContextDoc(link), '/**'))`
    - when using comparison operators with numbers or dates, it is not longer required that the datatypes match exactly, for example it is possible to compare a long with a decimal value.
    - various internal implementation cleanup and improvements.

- Field types:

    - Allow to use hierarchical selection lists with non-hierarchical fields. In this case, the selected node will be stored in the field, in contrast with hierarchical fields where the complete path leading to the node is stored.
    - Added a new kind of link-type hierarchical selection list; one where the tree is formed by documents having a link-type field pointing to their parent document.

- Repository API:

    - extracted the common methods of the Document and Version interfaces into a new interface called VersionedData.
    - QueryManager now provides an API to compile and evaluate arbitrary query-language expressions (previously there was only such an API for predicate expressions)
    - QueryHelper utility class: the patterns for formatting dates and times in the format accepted by the query language are now available as public strings.
    - The EvaluationContext.get/seUserId() methods have been removed. The user ID is now automatically taken from the current repository user.
    - When an entity (document, user, ...) cannot be updated due to optimistic locking, throw a ConcurrentUpdateException instead of a generic RepositoryException.

- Publisher:

    - extended p:forEach so that it can not only run over the results of a query, but also over the results of an expression if the expression returns a link-type result (thus a VariantKey). This is partly an alternative for the p:document/@field instruction (whose implementation is now changed to make use of the forEach), but allows more possibilities, e.g. in combination with the new GetLinkPath function.
    - the p:navigationTree instruction now allows to specify a p:document subrequest which, if present, will be executed for each document node in the navigation tree, the result is inserted as first child of the respective node. This allows to annotate the document nodes with extra information about the document.

  - Publisher exceptions now include a source location, in case of an exception the current execution stack in the publisher request is also shown.
  - In certain attributes or elements, it is now possible to use expressions using `${...}` syntax, where previously only fixed values where supported. This is for example useful with p:navigationTree to dynamically specify the navigation document or active document.
  - the p:performQuery instruction now allows to specify a p:document subrequest, similar as for p:navigationTree. The resulting functionality is similar to p:forEach, but gives you access to e.g. the chunk information.
  - allow faceted queries with the p:performFacetedQuery instruction.

- Navigation Manager:

  - nodes can now optionally be annotated with the number of children they have (both the number of normally-visible nodes and the number of all nodes, including the visible when-active or hidden nodes).
  - a link node can now be conditionally hidden depending on the read access to another document

- Document publishing:

  - the site's configured publisher request set is now also honored for extensions such as RSS feeds and publishdoc, and for the document basket aggregation.
  - a new property called displayContext is made available to the document styling XSLTs, next to the already existing isIncluded. displayContext can contain an arbitrary string value indicating the context in which the document is displayed. For the normal display of a document in the Wiki its value is "standalone", while in other cases it has other values or is empty. The displayContext is defined via the p:preparedDocuments instruction in the publisher request.

- Tanuki wrapper scripts:

  - included binaries for more platforms (Mac, Solaris), the binary to use is automatically determined.
  - don't copy the wrapper binaries to the data directories anymore, instead require the DAISY_HOME environment variable to be set
  - make use of wrapper's cascading configuration files feature so that only local customizations have to be in the data directories, avoiding the need to manually merge changes on upgrade.
  - For Windows: fixed failing repository shutdown by adding a dependency on the "Netman" service (see DSY-457[6] for details).
  - Added an extra DO_NOT_USE_THESE_SCRIPTS.txt file in the wrapper/service directory.

- Import/export tools:

  - The export-set can now specify extra schema types, namespaces and collections to be exported, that are not in used by the exported documents.
  - Field and part types used by documents, that are not part of their document type anymore, are now also exported.
  - import/export format change: there is a new info/collections.xml file

- Faceted browser :

  - accepts a set of additional identifiers which is used in select clause of the query.
  - supports ordering on names of link value types.

- Other, somewhat larger new functionality:

  - variables in documents (page 220)

  - Shifting headers for document includes: when including one document inside another (using an include or query-and-include), it is now possible to let the headers of the included document shift with a certain amount. This can be specified by locating the cursor inside the query instruction and using the "Include settings" button on the toolbar.
  - A first experimental version of the HTML diff by our Google Summer of Code student has been integrated.

- Other, somewhat smaller new functionality:

  - A per-site filter can now be defined for the document types that should be visible when creating a new document. This filter (using include/exclude patterns) is specified in the siteconf.xml (page 155)
  - Added a "Literal HTML (page 165)" document type allowing the embedding of any HTML.
  - The document editor now supports to plug in a "pre-save interaction" screen, an example of this is for eSignatures (page 0).
  - Allow to set the start number for ordered lists

- Other small improvements:

  - Below the navigation tree, there are now both "view" and "edit" links for the navigation document, displayed when the corresponding rights are available on the navigation tree.
  - Make the guest user configurable for the guest repository provider. This allows to change the password of the guest user.
  - daisy-util.js: added a getDaisy() function to get (a singleton instance of) the Daisy object, which is somewhat nicer than having to do "new Daisy()".
  - It is now possible to get PDFs for each version of a document. Previously, the PDF was always generated for the live/last version of the document.

    - The styling of the "Variant Not Found" page has been improved to show the available variants in a tabular layout.
    - PDF publishing (in the Wiki): switched to FOP version 0.93, which should fix various issues, such as footnotes running through the body text. It also supports new features such as "keep with next" (useful for headings).
    - Upgraded to Jetty 6.1.3 (from 5.1.10), both for the Jetty embedded in the repository server as the one included in the binary distribution to run the Daisy Wiki.

- Other small fixes:

    - The [edit] links for included documents now also show up in IE 6.
    - Admin console: don't show edit and delete links for the default language and the main branch, since these are not editable anyway.
    - Fixed bug in document type filtering through ACL (for display on "New document" page) when using the AND operator.
    - Windows bat files: paths ending on backslash (for DAISY_HOME, repo/wiki data dir) no longer cause problems (DSY-461).
    - Link extraction: include-links were not recognized if the include instruction contained a comment.
    - Book publishing: document includes in books were not working properly, due to a wrong namespace declaration being added to the first element of the included content.
    - Fulltext index: if the index is locked on repository startup, assume it is from an unclean shutdown and automatically unlock the index. A big warning is printed and an error is logged.
    - Book PDF publication: fixed the problem with embedding images when the wiki data directory was located in a path containing spaces.
    - Fixed all HTML cleaner issues reported in Jira.

2.10 2.0(.x) to 2.1 compatibility

2.10.1 Skin compatibility

- There's a new layout-common.css, imported by layout.css and layout-mini.css
- The error.xsl changed a bit.
- There's a new plain.css, which needs to be linked by the layout.xsl. If you have a custom layout.xsl, then update it similar to the new default.xsl (just search in there for plain.css)
- layout.xsl: implemented 'view/edit' link for navigation document: see NavigationDocActions template.

2.10.1.1 XSL-FO (stylesheets for PDF)

Daisy 2.1 ships with a major new release of the XSL-FO processor, FOP 0.93. If you have custom XSL-FO stylesheets, it could be there are smallish compatibility issues.

2.10.2 Repository extensions, authentication schemes, etc

2.10.2.1 New Runtime

Since we moved from Avalon Merlin to the new Daisy Runtime, you will need to adjust your repository extensions, authentication schemes, etc. to be compatible with the new infrastructure.

Some pointers to more information:

- General information on the Daisy Runtime (page 141).
- Tutorial on how to create an authentication scheme (page 137).
- The new sources (e.g. of the authentication schemes) can also serve as help.

If you have trouble adjusting your extensions or understanding the new system, you can ask questions on the Daisy mailing list.

2.10.2.2 Package move

Most of the SPI classes have been moved to different packages. For example:

```
org.outerj.daisy.authentication => org.outerj.daisy.authentication.spi
```

It should be easy to adjust your classes, which you'll need to do anyhow for the new Daisy Runtime.

2.10.2.3 AbstractAuthenticationFactory

This class is deprecated (and non-functional). If you used this, see the updated NTLM and LDAP authentication schemes for how to update your code.

2.10.3 Publisher wraps exception

The Publisher now wraps any exception occurring in the publisher with a GlobalPublisherException, containing information on the execution stack of the publisher. This might have effects on how you handle exceptions coming from the publisher. For example you might do a catch for GlobalPublisherException and then do getCause() on it to get the actual exception.

The error.xsl has also been changed to hide the GlobalPublisherException.

2.10.4 Book publisher

2.10.4.1 If you're using custom book publication types

The `shiftHeaders` task has been deprecated, the heading shifting is now performed as part of the `assembleBook` task. This change had to be made in order to implement the new heading shifting for document includes.

Normally, you don't need to adjust anything: the shiftHeaders task still exists but now does nothing at all. To avoid future confusion, it is recommended you remove the shiftHeaders task from any custom book publication types you might have.

2.10.5 Changes to non-public things

The following are changes to Daisy internals that might be relevant for some users.

2.10.5.1 Constants.DAISY_LINK_PATTERN

This is not really a part of the public API, but if you would happen to use the regex pattern defined in Constants.DAISY_LINK_PATTERN, you might have to adjust your code because the structure of this pattern has changed a bit: meaningless groups have been changed into non-capturing groups. See the javadoc of that constant for the exact matching groups.

2.10.5.2 Change to htmlcleaner.xml

The pre element now allows a daisy-shift-headings attribute for the new heading shifting for document includes feature.

2.10.6 Automated installation

When making use of the possibility to specify a property file to the repository-server-init script, two new properties are now required: dbName and jmsDbName, containing the names of the databases (= the same as those which are the JDBC URL).

2.11 2.0(.x) to 2.1. upgrade

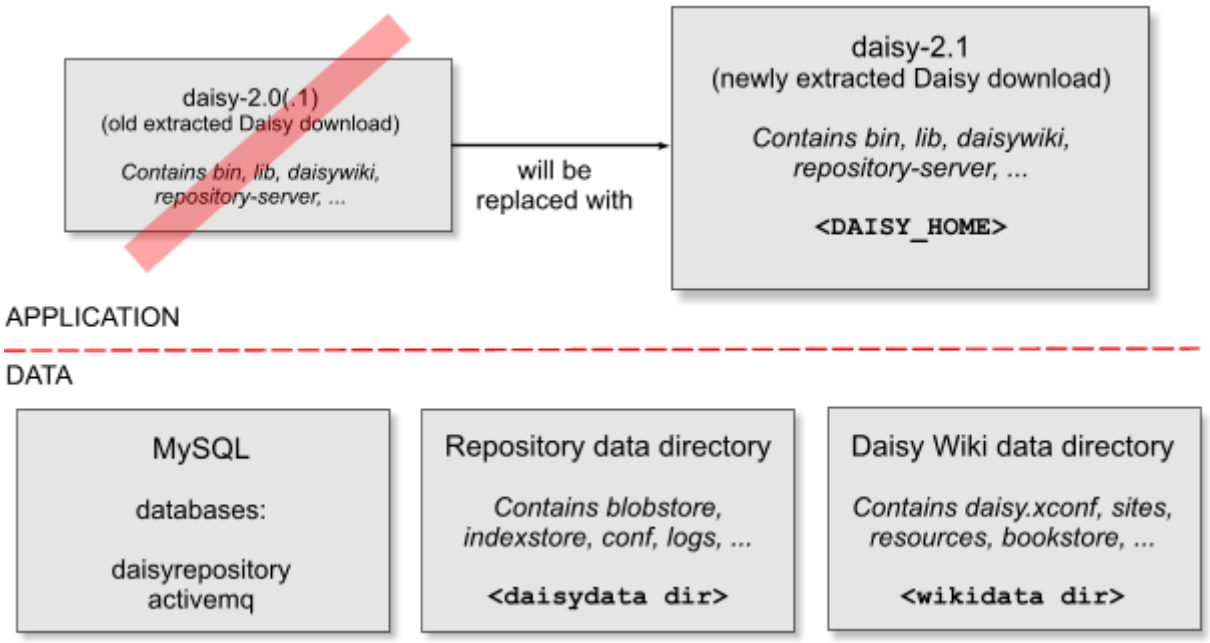
These are the upgrade instructions for when you have currently Daisy 2.0 or 2.0.1 installed.

If you have 2.1-RC installed, see here (page 40).

2.11.1 Upgrading

2.11.1.1 Daisy installation review

In case you're not very familiar with Daisy, it is helpful to identify the main parts involved. The following picture illustrates these.



There is the application directory, which is simply the extracted Daisy download, and doesn't contain any data (to be safe don't remove it yet though).

Next to this, there are 3 locations where data (and configuration) is stored: the relational database (MySQL), the repository data directory, and the wiki data directory. The Daisy repository and the Daisy Wiki are two independent applications, therefore each has its own data directory.

The text between the angle brackets (< and >) is the way we will refer to these directories further on in this document. Note that `<DAISY_HOME>` is the new extracted download (see later on), not the old one.

2.11.1.2 Stop your existing Daisy

Stop your existing Daisy, both the repository server and the Daisy Wiki.

2.11.1.3 Download and extract Daisy 2.1

If not done already, download Daisy 2.1 from the distribution area[7] (Sourceforge). For Windows, download the zip or autoextract.exe (*not the installer!*). For Unix-based systems, the .tar.gz is recommended. The difference is that the .zip contains text files with DOS line endings, while the .tar.gz contains text files with unix line endings. When using non-Linux unixes such as Solaris, be sure to use GNU tar to extract the archive.

Extract the download at a location of your choice. Extract it next to your existing Daisy installation, do not copy it over your existing installation.

2.11.1.4 Update environment variables

Make sure the DAISY_HOME environment variable points to the just-extracted Daisy 2.1 directory.

Note that when you start/stop Daisy using the wrapper scripts, you don't need to set `DAISY_HOME`, t hough you do need to update or re-generate the service wrapper configuration (see next section).

How this is done depends a bit on your system and personal preferences:

- it might be that you set the DAISY_HOME variable each time at the command prompt, in which case you simply continue to do this but let it now point to the new location:

```
[Windows]
set DAISY_HOME=c:\path\to\daisy-2.1
```

-

```
[Linux]
export DAISY_HOME=/path/to/daisy-2.1
```

- in Windows, it might be set globally via the System properties
- it could also be that you renamed the Daisy 2.0 directory to something that doesn't contain the version number (such as simply "daisy"), in which case you can leave the DAISY_HOME setting alone and just rename the daisy directories.

2.11.1.5 Creating log configuration

Daisy now uses log4j for logging, which needs a new configuration file in the repository data directory.

Therefore copy the file

```
<DAISY_HOME>/repository-server/conf/repository-log4j.properties
```

to

```
<REPO DATA DIR>/conf/
```

2.11.1.6 Updating the repository SQL database

Execute the database upgrade script:

```
cd <DAISY_HOME>/misc
mysql -Ddaisyrepository -udaisy -ppassword < daisy-2_0-to-2_1.sql
```

On many MySQL installations you can use "root" as user (thus specify `-uroot` instead of `-udaisy`) without password, thus without the `-p` option.

2.11.1.7 Adjusting the daisy.xconf file

Open the following file in a text editor:

```
<wiki data dir>/daisy.xconf
```

At the end of this file, before the closing </cocoon> tag, add these lines:

```
<component
    class="org.outerj.daisy.frontend.GuestRepositoryProviderImpl"
    role="org.outerj.daisy.frontend.GuestRepositoryProvider"
    logger="daisy">
    <guestUser login="guest" password="guest"/>
</component>
```

2.11.1.8 ActiveMQ configuration

The repository-server-init script of Daisy 2.0(.1) made an error in the ActiveMQ configuration. If you upgraded your 2.0 from earlier releases, the configuration should be OK, but there's no harm in checking it anyhow.

Open the following file in a text editor:

```
<daisydata dir>/conf/activemq-conf.xml
```

If you find the following line in that file, **remove** it:

```
<property name="poolPreparedStatements" value="true"/>
```

2.11.1.9 Jetty configuration (only when using a custom jetty-daisywiki.xml)

If you have a custom jetty-daisywiki.xml in your wikidata directory, it will need updating because Daisy 2.1 contains a major new Jetty version (6.1.3).

The easiest is probably to start from the new default jetty-daisywiki.xml found at

```
<DAISY_HOME>/daisywiki/conf/jetty-daisywiki.xml
```

and change what you want to change (usually just the HTTP port number).

The new default jetty-daisywiki.xml enables request logging by default. You might want to disable this if you have a webserver in front which also does request logging.

2.11.1.10 Wrapper scripts

This section is only applicable if you are using the wrapper scripts.

Various updates have been done to the wrapper scripts.

An important difference is that the wrapper scripts now require DAISY_HOME to be set.

Please see the wrapper documentation (page 301) on how to regenerate the service wrapper scripts.

2.11.1.11 Start the servers

Make sure the DAISY_HOME environment variable points to the new Daisy 2.1 directory (you might want to rename the old directory to avoid it is still used by accident).

2.11.1.12 Update the default repository schema

There are some new schema types, therefore update the repository schema by running the daisy-wiki-init script:

```
[Windows]
cd <DAISY_HOME>\install
daisy-wiki-init

[Linux]
```

```
cd <DAISY_HOME>/install
./daisy-wiki-init
```

2.12 2.1-RC to 2.1 upgrade

2.12.1 Changes since 2.1-RC

- Further improvements to the HTML diff and the diff page in general. A direct link Actions -> Changes has been added to go to the diff without having to go via the versions page.
- Upgraded to FOP 0.94 (unfortunately, the footnotes-in-tables-and-lists bug is not yet fixed).
- The unit px was not working for image print-sizes.
- Updated french translations.
- And a few minor things.

2.12.2 Upgrade instructions

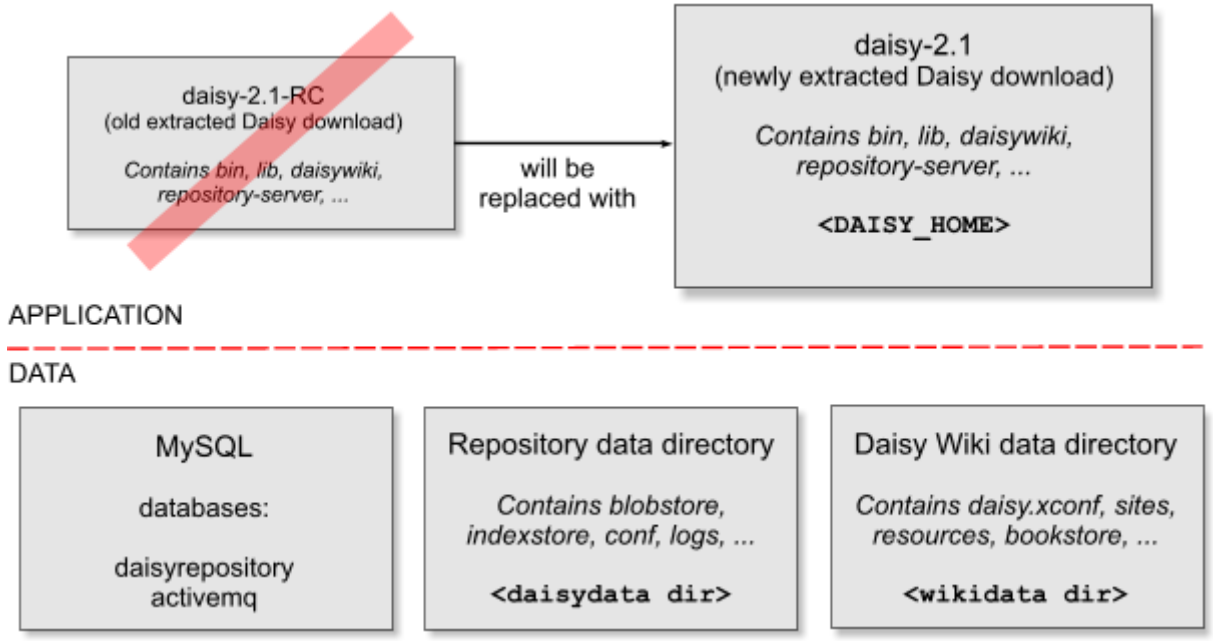
These are the upgrade instructions for when you have currently Daisy 2.1-RC installed.

This release requires no special upgrade steps, besides putting the new Daisy distribution in place.

In case you have problems during the upgrade or notice errors or shortcomings in the instructions below, please let us know on the Daisy mailing list.

2.12.2.1 Daisy installation review

In case you're not very familiar with Daisy, it is helpful to identify the main parts involved. The following picture illustrates these.



There is the application directory, which is simply the extracted Daisy download, and doesn't contain any data (to be safe don't remove it yet though).

Next to this, there are 3 locations where data (and configuration) is stored: the relational database (MySQL), the repository data directory, and the wiki data directory. The Daisy repository and the Daisy Wiki are two independent applications, therefore each has its own data directory.

The text between the angle brackets (< and >) is the way we will refer to these directories further on in this document. Note that `<DAISY_HOME>` is the new extracted download (see later on), not the old one.

2.12.2.2 Stop your existing Daisy

Stop your existing Daisy, both the repository server and the Daisy Wiki.

2.12.2.3 Download and extract Daisy 2.1

If not done already, download Daisy 2.1 from the distribution area[8] (Sourceforge). For Windows, download the zip or autoextract.exe (*not the installer!*). For Unix-based systems, the .tar.gz is recommended. The difference is that the .zip contains text files with DOS line endings, while the .tar.gz contains text files with unix line endings. When using non-Linux unixes such as Solaris, be sure to use GNU tar to extract the archive.

Extract the download at a location of your choice. Extract it next to your existing Daisy installation, do not copy it over your existing installation.

2.12.2.4 Update environment variables

Make sure the DAISY_HOME environment variable points to the just-extracted Daisy 2.1 directory.

Note that when you start/stop Daisy using the wrapper scripts, you don't need to set `DAISY_HOME`, though you do need to update or re-generate the service wrapper configuration (see next section).

How this is done depends a bit on your system and personal preferences:

- it might be that you set the DAISY_HOME variable each time at the command prompt, in which case you simply continue to do this but let it now point to the new location:

```
[Windows]
set DAISY_HOME=c:\path\to\daisy-2.1
```

-

```
[Linux]
export DAISY_HOME=/path/to/daisy-2.1
```

- in Windows, it might be set globally via the System properties
- it could also be that you renamed the Daisy 2.1-RC directory to something that doesn't contain the version number (such as simply "daisy"), in which case you can leave the DAISY_HOME setting alone and just rename the daisy directories.

2.12.2.5 Start Daisy

Start Daisy using the normal scripts (page 300) or the wrapper scripts (page 301).

Notes

1. http://sourceforge.net/project/showfiles.php?group_id=176692
2. http://lists.cocoondev.org/mailman/listinfo/daisy
3. http://java.sun.com/j2se/1.5.0/download.jsp
4. https://jai.dev.java.net/binary-builds.html#Release_builds
5. http://dev.mysql.com/downloads/
6. http://issues.cocoondev.org/browse/DSY-457
7. http://sourceforge.net/project/showfiles.php?group_id=176692
8. http://sourceforge.net/project/showfiles.php?group_id=176692

3 Source Code

Sources can be obtained through SVN[1]. Instructions for setting up a development environment with Daisy (which is slightly different from using the packaged version) are included in the README.txt's in the source tree. For anonymous, read-only access to Daisy SVN, use the following command:

```
svn co http://svn.cocoondev.org/repos/daisy/trunk/daisy
```

This will give the latest development code (the "trunk"). To get the source code of a specific release, use a command like this:

```
svn co http://svn.cocoondev.org/repos/daisy/tags/RELEASE_1_3_1 daisy
```

See also the existing tags[2].

No authentication is required for anonymous access. If you're behind a (transparent) proxy, you might want to verify whether your proxy supports the extended HTTP WebDAV methods[3].

3.1 Daisy Build System

We should consider removing this document, Maven is common enough these days.

The build system used by Daisy is Maven[4], an Apache project.

3.1.1 Maven intro

What follows is the very-very-quick Maven intro, for those not familiar with Maven.

Unlike Ant, where you tell how your code should be build, in Maven you simply tell what directory contains your code, and what the **dependencies** are (i.e. what other jars it depends on), and it will build your code. This information is stored in the **project.xml** files that you'll see across the Daisy source tree. There are a lot of them, since Daisy is actually composed of a whole lot of mini-projects, whereby some of these projects depend on one or more of the others.

An important concept of Maven is the **repository**, which is a repository of so-called **artifacts**, usually jar files. An artifact in the repository is identified uniquely by a group id and an id (both are simply descriptive names). Declaring the dependencies of a project is done by specifying repository references, thus for each dependency you specify the group id and id of the dependency. An example dependency declaration, as defined in a project.xml file:

```
    <dependency>
      <groupId>lucene</groupId>
      <artifactId>lucene</artifactId>
      <version>1.3</version>
    </dependency>
```

So where does the repository physically exist? Well, there can be many repositories. The most important public one is on ibiblio:

```
http://www.ibiblio.org/maven/
```

The repository is simply accessed using HTTP, so you can take your browser and surf to that URL. A repository like the one on ibiblio is called a **remote repository**. After initially downloading an artifact from the remote repository, it is installed in your **local repository**, which is by default located in ~/.maven/repository.

When you build a project, the result of the build is usually a jar file. Maven will install this jar file in your local repository, so that when you build another project that depends on this jar file, it can be found over there. When searching a dependency, Maven always checks the local repository first, and then goes off checking remote repositories. Which remote repositories are searched is of course configurable.

I should also tell you something about the **build.properties** and **project.properties** files. Both files contain properties for the build and configuration for Maven. The difference is that the project.properties files are committed to the source repository (SVN in Daisy's case), while the build.properties files are intended for local customisations (thus on your computer). So if you see something in a project.properties file that you'd like to change, don't change it over there (as this will otherwise show up as a modified file when doing svn status), but do it in the build.properties file. The build.properties file thus has a higher precedence than the project.properties file.

There is a lot more to tell about Maven, such as that it is actually composed of a whole lot of plugins, that there is something like "goals" to execute, that there is the possibility to have a maven.xml file to define custom goals with custom build instructions, and that all artifacts are also versioned. But I'll let you explore the Maven documentation to learn about that.

3.1.2 Extra dependencies

Daisy has some dependencies on artifacts (remember, jar files) that are not available in the public ibiblio repository. We make these available in our own repository on `http://cocoondev.org/repository/`.

3.1.3 Building Daisy

Instructions for building Daisy can be found in the README.txt file in the root of the Daisy source tree. At some point it will tell you to execute maven in the root of the source tree, which will actually build all the little mini-projects of which Daisy consists, in the correct sequence so that all dependencies are satisfied.

Notes

1. http://subversion.tigris.org/
2. http://svn.cocoondev.org/viewsvn/tags/?root=daisy
3. http://subversion.tigris.org/faq.html#proxy

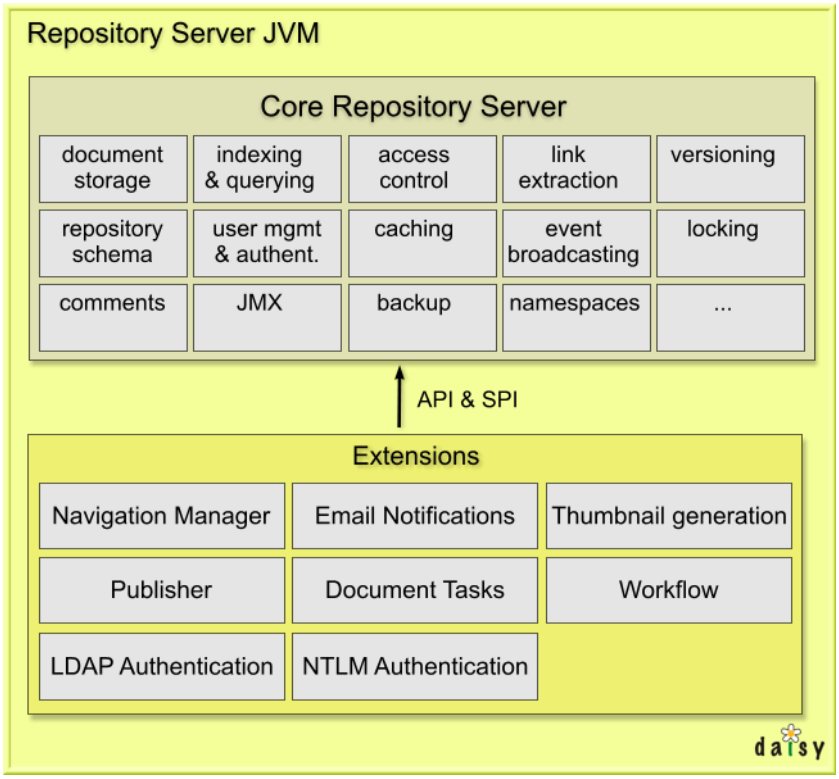
4. http://maven.apache.org/

4 Repository server

The repository server is the core of Daisy. It provides the pure content management functionality without GUI (graphical user interface).

The main purpose of the repository is managing documents (page 46).

The repository server consists of a core and some non-essential extension components that add additional functionality. The repository can be accessed by a variety of client applications (such as web applications, command-line tools, desktop applications, ...) through its programming interfaces (page 111).



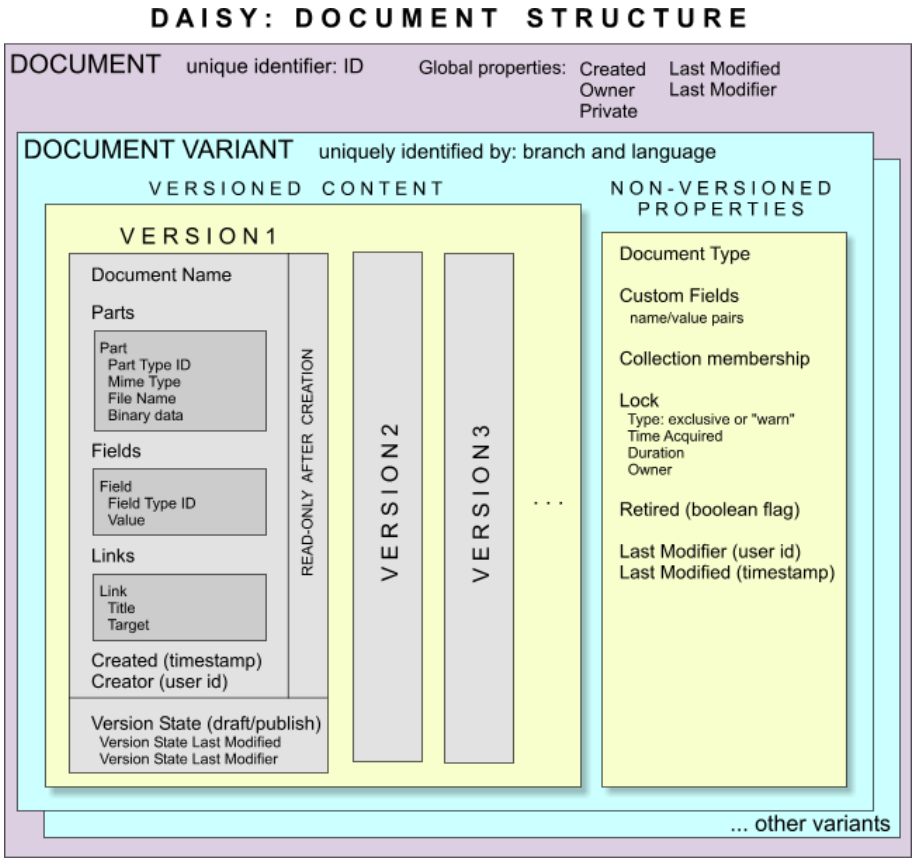
4.1 Documents

4.1.1 Introduction

The purpose of the Daisy Repository Server is managing documents. The main content of a document is contained in its so-called *parts* and *fields*. Parts contain arbitrary binary data (e.g. an

XML document, a PDF file, an image). Fields contain simple information of a certain data type (string, date, decimal, ...).

The diagram below gives an overview of the document structure, this is explained in more detail below.



4.1.2 No hierarchy

Daisy has no folders or directories like a filesystem, all documents are stored in one big bag. When saving a document, you only have to choose a name for it (which acts in fact as the title of the document), and this name is not even required to be unique (see below). Documents are retrieved by searching or browsing. Front-end applications like the Daisy Wiki allow to define multiple hierarchical views on the same set of repository documents.

4.1.3 Documents & document variants

A document can exist in multiple variants, e.g. in multiple languages. A document in itself does not consist of much, most of the data is contained in the document variants. From another point of view (which closer matches the implementation), one could say that the repository server actually manages document variants, which happen to share a few properties (most notably their identity) through the concept of a document.

A document has always at least one document variant, a document cannot exist by itself without variants.

A document is identified uniquely by its ID, a document variant is identified by the triple {document ID, branch, language}.

If you are not interested in using variants, you can mostly ignore them. In that case each document will always be associated with exactly one document variant. Therefore, **often when we speak about a document in Daisy, we implicitly mean "a certain variant of a document" (a "document variant").** In a practical working environment like the Daisy Wiki, the branch and language which identify the particular variant of the document are usually a given (Daisy Wiki: configured per site), and you'll only work with document IDs, so it is as if the existence of variants is transparent.

Refer to the diagram above to see if a certain aspect applies to a document, a document variant, or a version of a document variant.

For more details on this topic, see variants (page 57).

4.1.4 Document properties

4.1.4.1 ID

When a document is saved for the first time, it is assigned a unique ID. The ID is the combination of a sequence counter and the repository namespace (page 60). If the repository namespace is FOO, then the first document will get ID 1-FOO, the second 2-FOO, and so on. The ID of a document never changes.

4.1.4.2 Owner

The owner of a document is a person who is always able to access (read/write) the document, regardless of what the ACL specifies. The owner is initially the creator of the document, but can be changed afterwards.

4.1.4.3 Created

The date and time when the document was created. This value never changes.

4.1.4.4 Last Modified and Last Modifier

Each time a document is saved, the user performing the save operation is stored as the last modifier, and the date and time of the save operation as the "last modified" timestamp.

Note that each document variant has their own (page 52) last modified and last modifier properties, which are usually more interesting: the last modified and modifier of the document are only updated when some of the shared document properties change.

4.1.5 Document variant properties

4.1.5.1 Versions

A document consists of versioned and non-versioned data. Versioned data means that each time the document is saved (and some of the versioned aspects of the document changed), a new version will be stored, so that the older state of the data can still be viewed afterwards.

It hence provides a history of who made what changes at what time. It also allows to work on newer versions of a document while an older version stays the *live version*, as explained in version state (page 50).

4.1.5.2 Versioned Content

The versioned content of a document consists of the following:

- the document name
- the parts
- the fields
- the links

So if any changes are made to any of these, and the document is stored, a new version is created.

4.1.5.2.1 Version ID

Each version has an ID, which is simply a numeric sequence number: the first version has number 1, the next number 2, and so on.

4.1.5.2.2 Document Name

The name of a document is required (it cannot be empty). The name is not required to be unique. Thus there can be multiple documents with the same name. The ID of the document is its unique identification.

The name is usually rendered as the title of the document.

4.1.5.2.3 Parts

A part contains arbitrary binary data. "Binary data" simply means that it can be any sort of information, such as plain text, XML or HTML, an image, a PDF or OpenOffice document.

In contrast with many repositories or file systems, a Daisy document can contain multiple parts. This allows to store different types of data in one document (e.g. text and an image), and makes these parts separately retrievable.

For example, one could have a document with a part containing an abstract and a part containing the main text. It is then very easy and efficient to show a page with the abstracts of a set of document.

As another example, a document for an image could contain a part with the rendered image (e.g. as PNG), a part with a thumbnail image and a part with the source image file (e.g. a PhotoShop or SVG file).

The parts that can be added to a document are controlled by its document type (page 50).

Each part:

- is associated with a part type (page 54).
- has some binary data. There are no specific restrictions on the size of the data, Daisy handles everything using streams.
- has a mime-type, describing the sort of data stored in it.
- optionally has a file name, this file name can be used as default file name when the content of the part is saved (downloaded) in a file.

4.1.5.2.4 Fields

Fields contain simple information of a certain data type (string, date, decimal, ...). Depending on how you look at it, fields could be metadata about the data stored in the parts, or can be data by themselves.

One of the data types supported for fields is *link*, which allows the field to contain a link to another Daisy document. Link-type fields are useful for defining structured links (associations) between documents. For example, you could have documents describing wines, and other documents describing regions. Using a link-type field you can connect a wine to a region. By having this association in a field, it is easy to perform searches such as all wines associated with a certain region. The Daisy Wiki (page 155) allows, by means of the Publisher (page 90), to aggregate data from linked documents (page 183) when displaying a document, which combined with some custom styling allows to do very interesting things.

Fields can be *multi-valued*. The order of the values in a multi-value field is maintained. The same value can appear more than once.

A field can be *hierarchical*, meaning that its value represents a hierarchical path. A field can be multi-value and hierarchical at the same time.

The fields that can be added to a certain document are specified by its document type (page 50).

Each field:

- is associated with a field type (page 54).
- has a value.

4.1.5.2.5 Links

A document can contain links in the content of parts (for example, an `<a>` element in HTML) or in link-type fields. Next to this a document can have a number of so-called *out-of-line* links. These are links stored separately from the content. Each link consists of a title and a target (some URL). These links are usually rendered at the bottom of a page in as a bulleted list.

Out-of-line links are useful in case you want to link to related documents (or any URL) and either don't want or can't (e.g. in case of non-HTML content) link to them from the content of a part.

4.1.5.2.6 Version state & the live version

Each version can have a state indicating whether it is a draft version (i.e. you started editing the document but are not finished yet, in other words the changes should not yet be published), or a publishable version. The most recent version having the state 'publish' becomes the *live version*. The live version is the version that is typically shown by default to the user. It is also the version whose data is indexed in the full-text index, and whose properties are used by default when querying. The ACL (page 85) enables to restrict access for users to only the live versions of documents.

4.1.5.3 Non-versioned properties

4.1.5.3.1 Document type

Each document is associated with a document type, describing the parts and fields the document can contain. See repository schema (page 52) for more information on document types.

4.1.5.3.2 Collections and collection membership

Collections are sets of documents. A document can belong to zero, one or more collections, thus collections can overlap. A collection is simply a way to combine some documents in order to do something with them or treat them in some special way. In other words, they are a sort of built-in (always present) metadata to identify a set of documents.

Collections themselves can be created or deleted only by Administrators (in the Daisy Wiki, this is done in the administration interface). Deleting a collection does not delete the documents in it. You can limit who can put documents in a collection by ACL (page 85) rules.

4.1.5.3.3 Custom fields

Custom fields are arbitrary name-value pairs assigned to a document. The name and value are both strings. In contrast with the earlier-mentioned fields that are part of the document type, these fields are non-versioned. This makes it possible to stick tags to documents without causing a new version to be created, and without formally defining a field type.

4.1.5.3.4 Private

A document marked as private can only be read (and written) by its owner (page 48).

While the global access control system of Daisy makes it easy to centrally handle access control for sets of documents, sometimes it could be useful to simply say "I want nobody else to see this (for now)". This can be done by enabling the private flag. The document will then not be accessible for others, and also won't turn up in search results done by others. The private flag can be set on or off at any time, by the owner or by an Administrator.

There is however one big exception: Administrators can always access all documents, and thus will be able to read your "private" documents. The content is not encrypted.

4.1.5.3.5 Retired

If a document variant is no longer needed, because its content is outdated, replaced by others, or whatever, you can mark the document variant as retired. This makes the document variant virtually deleted. It won't show up in search results anymore.

The retired flag can be set on or off at any time, retiring is not a one-time operation.

4.1.5.3.6 Lock

A lock can be taken on a document variant to make sure nobody else edits the document variant while you're working on it.

Daisy automatically performs so-called optimistic locking, this means that if person A starts editing the document, and then person B starts editing the document, and then person A saves the document, and then person B tries to save the document, this last operation will fail because the document has changed since the time person B loaded it. This mechanism is always enabled, it is not needed to take an explicit lock.

A lock can then be taken to make others aware that you are editing the document. A lock can be of two types: an exclusive lock or a warn lock. An exclusive lock is pretty much as its name implies: it is a lock exclusively for the user who requested it, and avoids that any one else will be able to save the document until you release the lock. A warn lock isn't really a lock, it is just an

informational mechanism to let others know that someone else also started to edit the document, but it doesn't enforce anything. Anyone else can still at any time save the document or replace the lock with their own.

A lock can optionally have a certain duration, if the duration is expired, the lock is automatically removed.

For example, the Daisy Wiki application by default uses exclusive locks with a duration of 15 minutes, and automatically extends them as longs as the user continues editing.

A lock can be removed either by the person who created it, or by an Administrator.

4.1.5.3.7 Last Modified and Last Modifier

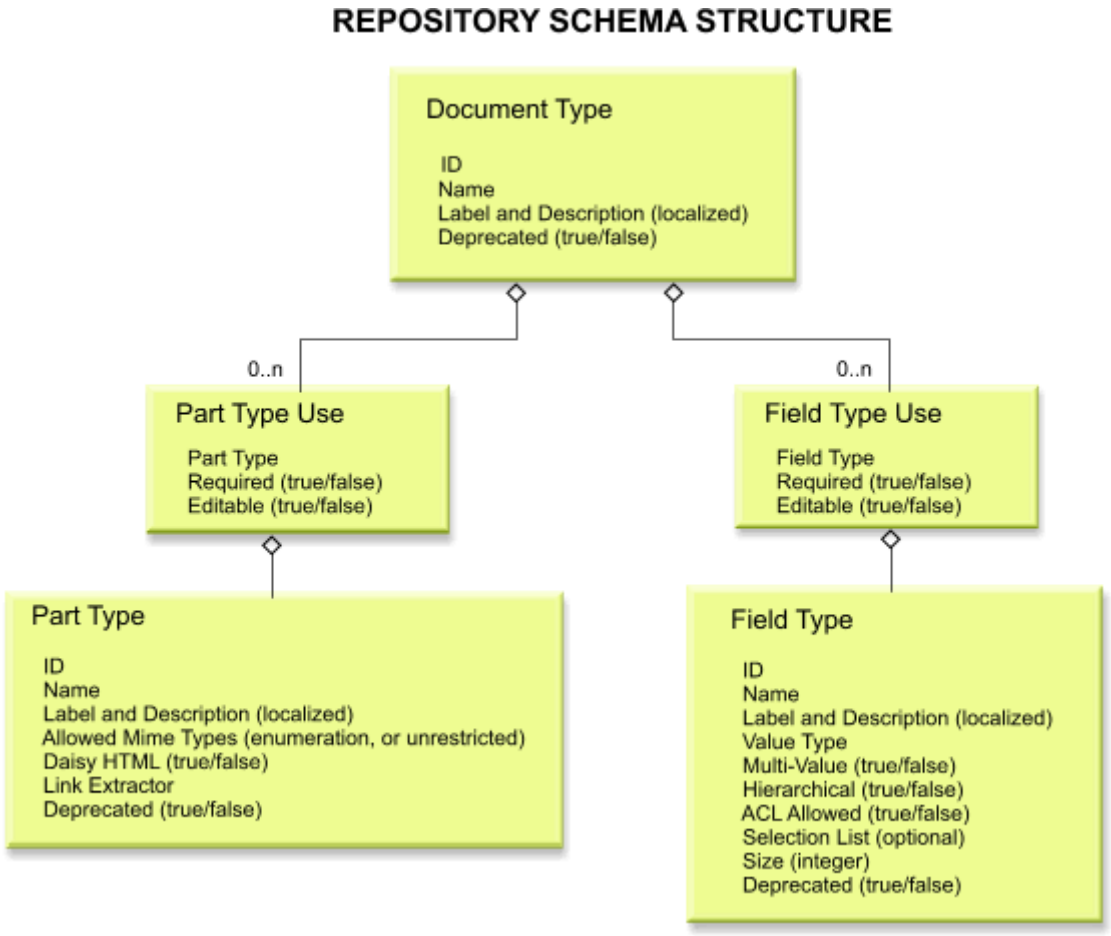
Each time a document is saved, the user performing the save operation is stored as the last modifier, and the date and time of the save operation as the "last modified" timestamp. This will often fall together with the Created/Creator fields of the last version, but not necessarily so: if only non-versioned properties are changed, no new version will be created.

4.2 Repository schema

4.2.1 Overview

The repository schema controls the structure of documents (page 46).

The repository schema defines part types, field types and document types. A document type is a combination of zero or more *part types* and zero or more *field types*. Part and field types are defined as independent entities, meaning that the same part and field types can be reused across different document types. The diagram below shows the structure and relation of all these entities.



4.2.1.1 Common aspects of document, part and field types

Let us first look at the things document, part and field types have in common. Their primary, unchangeable identifier is a numeric ID, though they also have a unique name (which can be changed after creation), which you will likely prefer to use.

Next to the name, they can be optionally assigned a localized label and a description. Localized means that a different label and description can be given for different locales. A locale can be a language, language-country, or language-country-variant specification. For example, a label entered for the locale "fr-BE " would mean it is in French, and specifically for Belgium. The labels and descriptions are retrieved using a fallback system. For example, if the user's locale is "fr-BE", the system will first check if a label is available for "fr-BE", if not found it will check for "fr", and finally for the empty locale "". Thus if you want to provide labels and descriptions but are not interested in localization, you can simply enter them for the empty locale.

Document, part and field types cannot be deleted as long as they are still in use in the repository. Once a document has been created that uses one of these types, the type can thus not be deleted anymore (unless the documents using them are deleted). However, it is possible to mark a type as *deprecated* to indicate it should not be used anymore. This deprecation flag is purely informational, the system simply stores it.

4.2.1.2 Document types

A document type combines a number of part types and field types. The association with the part and field types, in the diagram shown as the "Part Type Use" and "Field Type Use", are not stand-alone entities but part of the document type.

The associations have a property to indicate whether or not the parts and fields are required to have a value.

The associations also have a property called 'editable'. This property is a hint towards the document editing GUI that the part or field should not be editable. This is just a GUI hint, not an access control restriction. This can for example be useful if the values of certain fields or parts are assigned by an automated process.

4.2.1.3 Part types

A part type defines a part (page 49) that can be added to a document.

4.2.1.3.1 Mime-type

A part type allows to restrict which types of data (thus which mime-types) are stored in the part, but this is not required. This restriction is done by specifying a list of allowed mime types.

4.2.1.3.2 The Daisy HTML flag

A part type has a flag indicating whether the part contains "Daisy HTML". *Daisy HTML* is basically HTML formatted as well-formed XML (with element and attribute names lowercased). It is not the same as XHTML, because the elements are not in the XHTML namespace. If the "Daisy HTML" flag is set to true, the mime-type should be limited to `text/xml`. For the repository server, the Daisy-HTML flag on the part type has little meaning. Currently it serves only to enable the creation of document summaries (which might even be replaced with a more flexible mechanism in the future). The Daisy Wiki front end application will show a wysiwyg editor for Daisy HTML parts, and display the content of such parts inline.

4.2.1.3.3 Link extraction

For each part type a link extractor can be defined to extract links from the content contained in the part. The most common link extractor is the "daisy-html" one, which will extract links from the href attribute of the <a> element, the src attribute of the <img> element, and the character content of <p class="include">. The format of the links is:

```
daisy:<document id>
or
daisy:<document id>@<branch id or name>:<language id or name>:<version id>#fragment_id
```

Links that don't conform to this form will be ignored. The <version id> can take the special value "LAST" (case insensitive). A link without a version specification denotes a link to the live version of the document. The branch, language and version and fragment ID parts are all optional. For example, daisy:15@:nl is a link to the Dutch version of document 15.

The repository server also has link extractors for extracting links from navigation (page 166) and book (page 230) definition documents.

4.2.1.4 Field types

A field type defines a field (page 50) that can be added to a document.

4.2.1.4.1 Value Type

The most important thing a field type tells about a field is its *value type*. A value type identifies the kind of data that can be stored in a field, the available value types are listed in the table below, together with their matching Java class.

Value type name	Corresponding Java class
string	java.lang.String
date	java.util.Date
datetime	java.util.Date
long	java.lang.Long
double	java.lang.Double
decimal	java.math.BigDecimal
boolean	java.lang.Boolean
link	org.outerj.daisy.repository.VariantKey[1]

The link type is somewhat special: it defines a link to another document variant. Its value is thus a triple (document ID, branch ID, language ID). The branch ID and language ID are optional (value -1 in the VariantKey object) to denote they should default to the same as the containing document (in other words, the branch and language are relative to the document). The branch and language will usually be unspecified, since this allows copying content between the variants while the links stay relative to the actual variant.

4.2.1.4.2 Multi-value

The multi-value property of a field type indicates whether the fields of that type can have multiple values. All the values of a multi-value field should be of the same value type.

A multi-value field can have more than once the same value, and the order of values of a multi-value field is maintained. Thus the values of a multi-value field form an ordered list.

In the Java API, a multi-value value is represented as an Object[] array, in which the entries are objects of the type corresponding to the field's value type (e.g. an array of String's, or an array of Long's).

4.2.1.4.3 Hierarchical

The hierarchical property of a field type indicates that the value of the fields of that type is a hierarchical path (a path in some hierarchy). A path is often represented as a slash-separated string, e.g. Animals/Four-legged/Dogs.

Hierarchical fields are technically quite similar to multi-value fields, because a hierarchical path is also an ordered set of values. It is however possible for a field type to be both hierarchical and multi-value at the same time.

In the Java API, a hierarchical value is represented by a HierarchyPath[2] object:

```
org.outerj.daisy.repository.HierarchyPath
```

A multi-value hierarchical value is an array (Object[]) of HierarchyPath objects.

4.2.1.4.4 Selection Lists

It is possible to define a selection list for a field type. This is a list of possible values that an end user can choose from when completing the field. There are multiple available selection lists types:

- static selection list: manual enumeration of the selection list items. For each list item, you can specify the value, and optionally a label which will be shown to the user instead of the value. If desired, the label can be shown for different locales. If the static selection list belongs to a hierarchical field type, the static list can be hierarchical (each item can itself contain child items)
- query-based selection list: performs a query, typically selecting the value of some field, and takes the set of distinct values selected by the query as the content of the selection list.
- query-based selection list for link-type fields: similar to the query-based selection list, but since a link-type field points to some document, and a query returns a set of documents, it is not necessary to select a specific value of which the distinct set is taken. Rather the documents returned from the query are the content of the selection list.
- hierarchical childs-linked query selection list: this selection lists works by executing a query for the root values in the selection list, and then creates child items (the hierarchical items) by following specified (multi-value) link fields in the documents returned by the query.
- hierarchical parent-linked query selection lists: performs a query to retrieve documents and arranges them in a hierarchy based on link-field pointing to the parent of each document. Documents without the parent link-field become the first level in the hierarchy.

The hierarchical selection lists can be used both for hierarchical and non-hierarchical fields. For hierarchical fields, to whole path leading to the selected node is stored, for non-hierarchical fields only the selected node.

4.2.1.4.5 ACL allowed flag

In the access control (page 85) system, it is possible to define access rules for documents by using an expression to select the documents to which the access rules apply. In these expressions, it is also possible to check the value of fields, but only of fields whose field types' *ACL allowed flag* is set to true. The ACL allowed flag also enables the front-end to warn that changing the value of that particular field can influence the access control checks.

4.2.1.4.6 Size hint

A field can have a size hint, this is simply an integer number. This information is used by the front end to display an input field of an appropriate width. The repository server doesn't associate any further meaning to it, it doesn't cause any validation to happen, nor does it specify the unit of the width (most likely to be "number of characters").

4.2.1.5 Document and document type association, how changes to document types are handled

Upon creation of a document, a document type must be supplied. When saving a document, the repository will check that the document conforms to its document type. Thus it will check that all

required fields and parts are present, and that there are no parts and fields in the document that are not allowed by the document type.

The document type of a document can be changed at any time. This is useful if you start out with a generic document type but later want to switch to a more specialized document type.

The definition of a document type can be changed at any time. Part and field types can be added or removed from it, or can be made required. A logical question that pops up is what happens to existing documents in the repository that use the changed document type. The answer is basically "nothing". If for example a required field is added to a document type, then the next time a document of that type is edited, it will fail to save unless a value for the field is specified. The newly saved version of the document will then conform to the new state of the document type. Older versions of the document will remain unchanged however. When saving a document, it is also possible to supply an option that tells not to do the document type conformance check.

So basically the document type system doesn't give any guarantees about the structure of the documents in the repository, but rather hints at how the documents should be structured and interpreted.

See also the FAQ entry How do I change the document type of a set of documents? (page 0)

4.3 Variants

4.3.1 Introduction

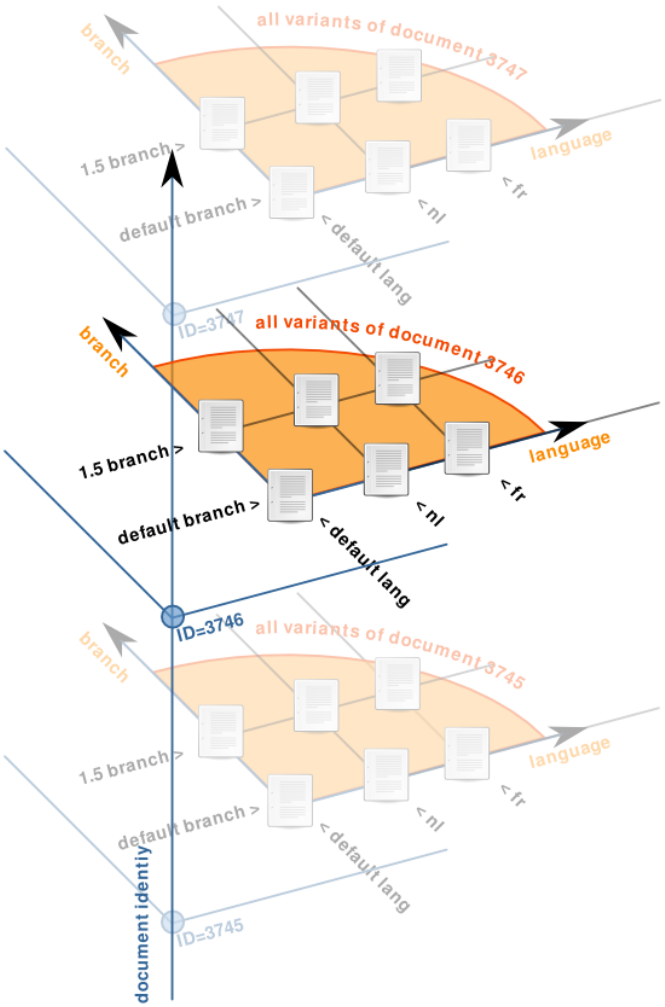
The variants feature of Daisy allows to have multiple alternatives of a document stored in one logical document, thus identified by one unique ID.

Daisy allows to have variants among two axes:

- branches
- languages

For example, if there would not be a variants feature, and you had the same content in different languages, for each of these languages you would need to create a different document, thus with a different ID.

Language variants are quite obvious, but you may wonder what branches are. The purpose of branches is to have multiple parallel editable versions of the same content. As an example, take the Daisy documentation. Between major Daisy releases there might be quite some changes to the documentation. However, while creating the documentation of e.g. Daisy 1.3, we still want the ability to update the documentation of Daisy 1.2. Sure, this could be solved by duplicating all documentation documents for each new release, but then the identity of these documents would be lost since they get new IDs assigned, and the relationship between the documents in different releases would be lost.



4.3.2 Defining variants

By default, Daisy predefines one branch and one language variant: the branch *main* and the language *default*.

You can yourself define other ones, in the Daisy Wiki you can do this via the administration screens.

The definition of a branch or language consists of a numeric ID (assigned by the repository server), a name and optionally a description. Internally, the ID is used, but towards the user mostly the name is shown.

The built-in *main* branch and *default* language each have as ID 1.

Once a branch and/or language is defined, you can create new document variants using them.

Defining the branches and languages is something that can only be done by users who have the Administrator role, but adding variants to documents (which is almost the same as creating documents) can of course be done by any user, as far as the ACL allows the user to do so.

Deleting a branch or language definition is only possible when there are no more document variants for that branch or language. You can easily delete all document variants for a certain branch or language using the Document Task Manager, similarly to what is described further on for creating a variant across a set of documents.

4.3.3 Creating a variant on a document

When adding a new variant to a document, this can be done in two ways:

1. from scratch
2. based on the content of (a certain version of) an existing variant

When you opt for the second option (which is mostly done when creating branch-variants) then the (branch,language,version)-triple from which the content is taken will be stored as part of the new variant, so that later on you can see from where this variant "branched" (in the Daisy Wiki, this information is shown on the version list page).

In the Daisy Wiki, there is an "Add Variant" action that allows to add a new variant to a document.

4.3.4 Searching for non-existing variants

When translating a site, it can be useful to search which documents are not yet translated in a certain language. Similarly, it can be useful to see which documents exist on one branch but not on another. For this purpose, the query language provides a function called `DoesNotHaveVariant(branch, language)`.

For example, to search on the Daisy site for all documents that have been added in the documentation of version 1.3 compared to 1.2, you can use the following query:

```
select id, name
  where
   InCollection('daisydocs')
   and branch = 'daisydocs-1_3' and language = 'en'
   and DoesNotHaveVariant('daisydocs-1_2', 'en')
```

4.3.5 Queries embedded in documents

When using queries embedded in documents together with variants, usually you will want to limit the query results to variants with the same branch and language as the one containing the query. You could specify these explicitly, as in:

```
select id, name where <conditions> and branch='my_branch' and language='my_lang'
```

However, this means that you will need to adjust these queries when adding new variants to the document. Especially if you are adding a certain branch to a set of documents, this is not something you want to do. Therefore, it should be possible to refer to the branch and language of the containing document. This can be done as follows:

```
select id, name where <conditions> and branchId = ContextDoc(branchId) and languageId =
ContextDoc(languageId)
```

4.3.6 Creating a variant across a set of documents

When using branches, you will often want to add a variant for that branch to a set of documents (in other words: create a branch across a set of documents). To avoid the need to do this one-by-one for each document, Daisy has a "Document Task Manager" which allows the

execution of a certain task on a set of documents. And that task could for example be "adding a new variant".

The Document Task Manager is covered in a separate section (page 90), here we will just focus on how to use it to create a new variant.

Before using the Document Task Manager, be sure you have defined the new branch (or language) using the administration screens.

In the Daisy Wiki, the Document Task Manager is accessed via the drop-down User-menu (in the main navigation bar). Select the option to create a new task. You are then first presented with a screen where you need to specify the documents (document variants actually) with which you want to do something. As you can see, it is possible to add documents using queries. For example, for the Daisy site, when we want to create a branch starting from the Daisy 1.2 documentation, we would use a query like:

```
select id, name where InCollection('daisydocs') and branch = 'daisydocs-1_2' and
language = 'en'
```

Once you selected the documents, press *Next* to go to the next page where the action to be performed on the documents is specified. For *Type of task* choose *Simple Actions*. Then press the *Add* button to add a new action. Change the type of the action to *Create Variant* (if necessary), and specify the branch and language you want to create. Finally press start to start the task. You can then follow up on the progress of this operation, and check if it finished successfully for all documents.

4.4 Repository namespaces

Each Daisy repository (since Daisy 2.0) has a namespace. The documents created in that repository will by default belong to that namespace. The ID of a document is the combination of a numeric sequence and the namespace, for example:

2583-AWAN

Each repository server is responsible for maintaining the sequence number for the documents of their namespace. When a document is created with a foreign namespace (a namespace from another repository server), the sequence ID needs to be supplied, since it is assumed that another repository server is responsible for that namespace. On the other hand, when a document is created in the namespace of the repository server itself, then a sequence ID cannot be supplied, as the repository server itself is then responsible for assigning the sequence ID.

A Daisy repository server doesn't really care whether foreign namespaces are really associated with other repository servers. The namespace could also come from an external application that creates documents (and maintains its own sequence counter), or it could come from a manually created 'export' that is imported using the import tool. So for foreign namespaces, it just assumes someone else is responsible for assigning unique sequence numbers.

4.4.1 Namespace name

A namespace can be up to 200 characters and contain the characters a-z, A-Z, 0-9 and _ (underscore). The dot character is not allowed to avoid confusion with file name extensions (as

the document ID will often be used in URLs), and the dash character is not allowed to avoid confusion with the separator between the sequence number and the namespace.

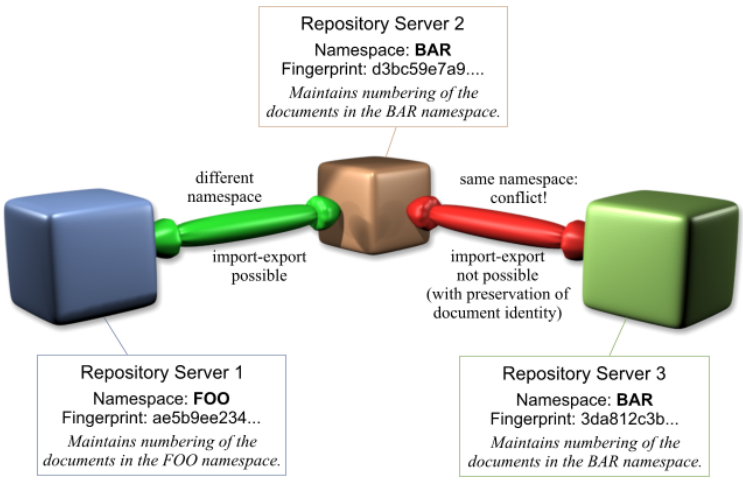
The namespace name is typically a short string. This approach has been choosen to keep the document IDs short and readable. But as a consequence, it requires some care to avoid conflicting namespaces within an organisation, and between organisations there's no control whatsoever.

If you want to avoid the possibilities of conflicts and don't care about the readability, it is always possible to use a longer string. For example, a registered domain name or a random generated string (GUID). However, in general we would recommend to stick with a short name.

The repository server installation tool gives some recommendations on namespace naming.

4.4.2 Namespace purpose

Namespaces have little purpose as long as documents are not exchanged between repositories. The main purpose of namespaces it to allow import/export (replication) of documents between repositories. If there wouldn't be namespacess, the document IDs between the repositories would not be unique and hence there would be conflicts. For example, in both repositories there might be a document with ID 55, though these would be different documents. It would of course be possible to assign new IDs to documents upon import, but then the identity of the original document would be lost, which would make subsequent 'update' imports difficult, and also requires updating links in all document content (which would mean the import tool has to understand the document formats).



4.4.3 Namespace fingerprints

For each namespace, there is an associated namespace fingerprint. Since namespaces will usually be short strings, and since there might be people who choose the same namespace or simply used the proposed default (DSY), some additional verification is needed to assure that two namespaces are really the same. For this purpose, each namespace name is associated with a fingerprint. The namespace fingerprint is typically a longer random generated string.

The repository server keeps a table of the namespaces used in the repository and their corresponding fingerprints (these are the namespaces that are said to be *registered* in the repository). The registered namespace are viewable through the administration screen of the Daisy Wiki, which also allows unregistering unused namespaces.

For export/import, namespace fingerprint information is included in the export, so that it can be verified upon import.

4.4.4 Namespacing of non-document entities

At the time of this writing, only documents are namespaced. Thus other entities, like the document, part and field types are not in a namespace. The export/import tools assume that if the name corresponds, they are the same.

4.5 Document Comments

This section is about document comments: comments that can be added to Daisy documents. More precisely, they are actually added to document variants, thus each variant of a document has its own comments.

4.5.1 Comment features

The current Daisy comments system is rather simple (text-only comments, no editing after creation, no threading) but nonetheless very useful.

4.5.1.1 Comment visibility

Each comment has a certain visibility:

- public comments: everyone who can read the document ('read live' permission) can see them,
- editors-only comments: only users who have write access to the document can see them,
- private comments: only the creator of the comment can see them.

4.5.1.2 Creation of comments

Everyone who has read access to a document can add comments to it. Editors-only comments can however only be created by users with write access to the document.

4.5.1.3 Deletion of comments

Comments can be removed from a document by the users who have write access to the document (this includes users acting in the Administrator role). Private comments can be deleted by its creator, independent of whether that user has write access to the document the comment belongs too.

When a document is deleted, all its associated comments are removed too, including private ones that the deleter of the document may not be aware of.

4.5.2 Daisy Wiki specific notes

4.5.2.1 Guest user cannot create comments

The guest user, though it is for the repository server an ordinary user like any other, is not allowed to create comments via the Daisy Wiki. This means that to create comments, users should first log in.

4.5.2.2 'My Comments' page

Users can get a list of all the private comments they added to documents via a "My Comments" page (accessible via the drop-down menu behind the user name).

4.6 Query Language

4.6.1 Introduction

The Daisy Query Language can be used to search for documents (more precisely, document variants). In the Daisy Wiki, queries can be used in various places:

- explicitly via the "Query Search" page
- embedded inside documents
- embedded inside navigation trees

The implementation of various Daisy Wiki features is also based on queries, such as the recent changes page or the referrers page. And of course it is possible to execute queries from your own applications, using the HTTP interface or Java API.

The query language is a somewhat SQL-like language that allows to search on various document properties (including the fields), fulltext on the part content, or a combination of those. The sort order of the results can also be defined. The resulting document list is filtered to only include documents to which the user has at least read-live access.

An example query, searching all documents in a collection call "mycollection":

```
select id, name where InCollection('mycollection') order by name
```

Internally, non-fulltext queries are translated to SQL and executed on the relational database while fulltext queries are executed by Jakarta Lucene[3].

Although the query language is somewhat SQL-like, it hides the complexity of the actual SQL-queries that are performed by the repository server on the relational database, which can quickly grow quite complex.

Note: every time in this document when we talk about "searching documents", this is equivalent to "searching document variants". The result of query is a set of document variants, i.e. each member of the result set is identified by a triple (document ID, branch, language).

4.6.2 Query Language

4.6.2.1 General structure of a query

```
select
  ...
where
  ...
order by
  ...
limit x
option
  ...
```

The `select` and `where` parts are required, the rest is optional. Whitespace is of no importance.

4.6.2.2 The select part

The `select` part should list one or more value expressions, separated by commas. A value expression can be an identifier, a literal or a function call. This is described in more detail further on.

4.6.2.3 The where part

The `where` part should contain a predicate expression, thus an expression which tests the value of *value expressions* using operators, or uses some built-in conditions.

Besides the operators listed in the table below, the operations `AND` and `OR` are supported, and parentheses can be used for grouping.

4.6.2.3.1 Operators & data types

	string	**long**	**double**	**decimal**	**date**	**datetime**	**boolean**
=	X	X	X	X	X	X	X
!=	X	X	X	X	X	X	X
<	X	X	X	X	X	X	
>	X	X	X	X	X	X	
<=	X	X	X	X	X	X	
>=	X	X	X	X	X	X	
[NOT] LIKE	X						
[NOT] BETWEEN	X	X	X	X	X	X	
[NOT] IN	X	X	X	X	X	X	
IS [NOT] NULL	X	X	X	X	X	X	X

Wildcards for `LIKE` are _ and %, escape using _ and \%.

All keywords such as `AND`, `LIKE`, `BETWEEN`, ... can be written in either uppercase or lowercase (but not mixed case).

If these operators are used on multi-value fields, they return true if at least one of the values of the multi-value field satisfies. See further on for a set of conditions specifically for multi-value fields.

Normally the comparison operators work on values of the same type, though there is some relaxation for compatible types, e.g. it is possible to compare between all numeric types, and between the date and datetime types.

4.6.2.4 Value expressions

A value expression is:

- an identifier (= some property of a document, see list further on)
- a literal (= a fixed value such as a string, a number or a date)
- a function call (whose arguments can be identifiers, literals or function calls)

A function call usually has the following form:

```
functionName(arg1, arg2, ...)
```

However, for the basic mathematical functions (addition, subtraction, multiplication and division) "infix" notation is used instead, using the symbols `+`, `-`, `*` and `/`. Parentheses can be used to influence the order of the operations.

4.6.2.5 Identifiers

The table below lists the available identifiers.

Some notes:

- identifier names are case sensitive
- non-searchable identifiers are identifiers which can only be used in the select clause of the query, not in the where clause
- the datatype symbolic means it should be a string, but the string is internally translated into another code. For example, when searching on ownerLogin, the given string is internally translated to a user id, which is then used when performing the database search. This means that certain operators will not work on it or will be of little meaning (such as like, less then, greater then, ...)
- version dependent means that the searched or retrieved data is version dependent data. By default this will search in, or retrieve data from, the live version of the document, but by specifying the query option `search_last_version` (see further on) the last version can also be searched.
- the names in italic, i.e. partTypeName, fieldTypeName and customFieldName must be replaced by an actual name.

name	**searchable**	**datatype**	**version dependent**	**remarks**
id	yes	string	no	

namespace	yes	string	no	The namespace part of the document ID
name	yes	string	yes	
branch	yes	symbolic	no	
branchId	yes	long	no	
language	yes	symbolic	no	
languageId	yes	long	no	
link	yes	link	no	The current document variant as a link. This is useful for comparison with link type fields. For example `$someLinkField = link` to find documents which link to themselves in a certain field, or `$someLinkField = ContextDoc(link)` to find documents which link to the context document.
documentType	yes	symbolic	no	
versionId	yes	long	yes	ID of the live version, or if the query option `search_last_version` is specified, of the last version
creationTime	yes	datetime	no	
ownerId	yes	long	no	
ownerLogin	yes	symbolic	no	
ownerName	no	string	no	
summary	no	string	no	always of last published version
retired	yes	boolean	no	
private	yes	boolean	no	
lastModified	yes	datetime	no	
lastModifierId	yes	long	no	
lastModifierLogin	yes	symbolic	no	
lastModifierName	no	string	no	
variantLastModified	yes	datetime	no	
variantLastModifierId	yes	long	no	
variantLastModifierLogin	yes	symbolic	no	
variantLastModifierName	yes	string	no	

%*partTypeName* .mimeType	yes	string	yes	
%*partTypeName*.size	yes	long	yes	
%*partTypeName*.content	no	xml	yes	only works for part types for which the flag 'daisy html' is set to true, and additionally the actual part must have the mime type 'text/xml'
versionCreationTime	yes	datetime	yes	
versionCreatorId	yes	long	yes	
versionCreatorLogin	yes	symbolic	yes	
versionCreatorName	yes	string	yes	
versionState	yes	symbolic	yes	'draft' or 'publish'
totalSizeOfParts	yes	long	yes	sum of the size of all parts in document
versionStateLastModified	yes	datetime	yes	
lockType	yes	symbolic	no	'pessimistic' or 'warn'
lockTimeAcquired	yes	datetime	no	
lockDuration	yes	long	no	(in milliseconds)
lockOwnerId	yes	long	no	
lockOwnerLogin	yes	symbolic	no	
lockOwnerName	no	string	no	
collections	yes	symbolic	no	The collections (the names of the collections) the document belongs too. Behaves the same as a multi-value field with respect to applicable search conditions.
collections.valueCount	yes	symbolic	no	The number of collections a document belongs too.
$*fieldTypeName*	yes		yes	datatype depends on field type
$*fieldTypeName* .valueCount	yes	long	yes	Useful for multi-value fields. Searching for a value count of 0 does not work, use the "is null" condition instead.
$*fieldTypeName* .documentId	yes	string	yes	These special field sub-identifiers are only supported on fields of the type "link". For link field types, the $*fieldTypeName* identifier checks on the document ID, while these identifiers can be used to check on the branch and language.
$*fieldTypeName*.branch	yes	symbolic	yes	

$*fieldTypeName*.branchId	yes	long	yes	
$*fieldTypeName*.language	yes	symbolic	yes	
$*fieldTypeName* .languageId	yes	long	yes	
$fieldTypeName .namespace	yes	string	yes	
#customFieldName	yes	string	no	
score	no	double	no	The score of a document after doing a full text search. This score ranges from 0-1. When this identifier is used without the FullText() function it will just return 0.

4.6.2.5.1 Addressing components of multivalue and hierarchical field identifiers

For multivalue and hierarchical field identifiers, an index-notation is supported using square brackets.

For multivalue fields:

```
$SomeField[index]
```

For multivalue hierarchical fields:

```
$SomeField[index][index]
```

For non-multivalue hierarchical fields, or if you only want to specify an index for the hierarchical value, you can use:

```
$SomeField[*][index]
```

The index is 1-based. You can address elements starting from the end by using negative indexes, e.g. -1 for the last element in a multivalue or hierarchy path.

In case you are using a sub-field identifier, it should be put after the square brackets:

```
$SomeField[index].documentId
```

Specifying an out-of-range index doesn't give an error, but simply finds/returns nothing.

4.6.2.6 Literals

4.6.2.6.1 String literals

Strings (text) should be put between single quotes, the single quote is escaped by doubling it, for example:

```
'''t is mooi weer vandaag'
```

4.6.2.6.2 Numeric literals

These consists of digits (0-9), the decimal separator is a dot (.).

Numeric literals can be put between single quotes like strings, but it is not required to do so.

4.6.2.6.3 Date & datetime literals

Date format: 'YYYY-MM-DD'

Datetime format: 'YYYY-MM-DD HH:MM:SS'

4.6.2.6.4 Link literals

When searching on fields of type "link", the link should be specified as:

```
'daisy:docid'               (assumes branch main and language default)
'daisy:docid@branch'        (assumes language default)
'daisy:docid@branch:lang'   (branch can be left blank which defaults to main branch)
```

Branch and language can be specified either by name or ID.

So a search condition could be for example:

```
$someLinkField = 'daisy:35'
```

4.6.2.7 Special conditions for multi-value fields

```
$fieldName has all (value1, value2, value3, ...)
```

Tests that the multi-value field has all the specified values (and possibly more).

```
$fieldName has exactly (value1, value2, value3, ...)
```

Tests that the multi-value field has all the specified values, and none more. The order is not important.

```
$fieldName has some (value1, value2, value3, ...)
or
$fieldName has any (value1, value2, value3, ...)
```

`has some` and `has any` are synonyms. They test that the multi-value field has at least one of the specified values.

```
$fieldName has none (value1, value2, value3, ...)
```

Tests that the multi-value field has none of the specified values.

In addition to these conditions, you can use `is null` and `is not null` to check if a document has a certain (multi-value) field. The special sub-identifier $*fieldName*.valueCount can be used to check the number of values a multi-value field has.

4.6.2.8 Searching on hierarchical fields

4.6.2.8.1 matchesPath

For searching on hierarchical fields, a special `matchesPath` condition is available. It takes as argument an expression in which the elements of the hierarchical path are separated by a slash. For example, a basic usage is:

```
$fieldName matchesPath('/A/B/C')
$fieldName matchesPath('A/B/C')  -> the initial slash is optional
```

This would return all documents for which the hierarchical field has as value the path A/B/C.

The values should be entered using the correct literal syntax corresponding to the type of the field. For example, for link type fields, you would use:

```
matchesPath('daisy:10-FOO/daisy:11-FOO')
```

It is possible to use wildcards (placeholders) in the expression, namely * and **. One stars (*) matches one path part. Two stars (**) matches multiple path parts. Two stars can only be used at the very start or at the very end of the expression (not at both ends at the same time). Some examples to give an idea of what's possible:

```
$fieldName matchesPath('/A/*')
$fieldName matchesPath('/A/*/*')
$fieldName matchesPath('/A/*/B')
$fieldName matchesPath('/*/*/*')  -> finds all hierarchical paths of length 3
$fieldName matchesPath('/*/*/**') -> finds all hierarchical paths of at least length 3
$fieldName matchesPath('/A/**')   -> finds all paths of any lenght starting on A
                                     thus e.g. A/B, A/B/C or A/B/C/D.
$fieldName matchesPath('**/A')    -> finds all paths ending on A
```

The argument of matchesPath should be a string, but doesn't have to be a literal. Some examples:

```
$fieldName matchesPath($anotherField, '/**'))

$fieldName matchesPath(Concat(ContextDoc(link), '/**'))

An example taken from Daisy's own knowledge base:
$fieldName matchesPath(String(ReversePath(
              GetLinkPath('KnowlegeBaseCategoryParent', 'true', ContextDoc(link)))))
```

4.6.2.8.2 Multi-value hierarchical fields

The `matchesPath` condition can also be used to search on multi-value hierarchical fields, in which case it will evaluate to true if at least one of the values of the multi-value field matches the path expression.

The special multi-value conditions such as 'has all', 'has some', etc. can also be used. There is no special syntax to specify hierarchical path literals in the query language, but they can be entered by using the `Path` function. For example:

```
$fieldName has all ( Path('/A/B/C'), Path('/X/Y/Z') )
```

The hierarchical paths specified using the Path function do not support wildcards.

4.6.2.8.3 Equals operator

When using the equals operator (=) or other binary operators with hierarchical fields, it will evaluate to true as long as there is one element in the hierarchy path which has the given value. For example, $MyField = 'b' will match a field whose value is "/a/b/c". This is similar to the behaviour of this operator for multivalue fields.

4.6.2.9 Link dereferencing

When an expression returns a link as value (most often this is in the form a link field identifier, e.g. $SomeLinkField), then it is possible to 'walk through' this link to access properties of the linked-to document. This is known as link dereferencing.

The link dereferencing operator is written as "=>". Notations for dereferencing in other languages are sometimes dot (.) or "->", however since dash is a valid character in identifiers in Daisy, and dot is already used to access 'sub-field identifiers' (like #SomePart.mimeType), these could not be used.

```
[link expression]=>[identifier]
```

A practical example:

```
select name, $SomeLinkField=>name where $SomeLinkField=>name like 'A%' order by
$SomeLinkField=>name
```

As shown in this example, the link dereferencing operator works in the select, where and order by parts of the query.

Link dereferencing can work multiple levels deep, e.g.

```
$SomeLink=>$SomeOtherLink=>name
```

If documents are linked together with the same type of field, this could of course be something like:

```
$SomeLink=>$SomeLink=>$SomeLink=>name
```

When dereferencing a link in the where-clause of the query, but one does not have access to the dereferenced document, then the evaluation of the where clause will be considered as 'false', e.g. the row will be excluded from the result set, since without access to the document it is not possible to know if it would evaluate to 'true'. Accessing non-accessible values in the select or order-by clauses will return a 'null' value.

4.6.2.10 Other special conditions

4.6.2.10.1 InCollection

```
InCollection('collectionname' [, collectionname, collectionname])
```

Searches documents contained in at least one of the specified collections. To search documents that occur in multiple collections (thus in the intersection of those collections), use the function InCollection multiple times with `AND` in between: `InCollection('collection1') and InCollection('collection2')`. This also works for `OR` but in that case it is more efficient to give the collections as arguments to one `InCollection` call.

Instead of the `InCollection` condition, you can use the `collections` identifier in combination with the multi-value field search conditions such as `has some`, `has all` or `has none` for more powerful search possibilities. The `InCollection` condition predates the existence of multi-value fields, but remains supported.

4.6.2.10.2 LinksTo, LinksFrom, LinksToVariant, LinksFromVariant

```
LinksTo(documentId, inLastVersion, inLiveVersion [, linktypes])
LinksFrom(documentId, inLastVersion, inLiveVersion [, linktypes])
LinksToVariant(documentId, branch, language, inLastVersion, inLiveVersion [, linktypes])
LinksFromVariant(documentId, branch, language, inLastVersion, inLiveVersion [, linktypes
])
```

Searches documents which link to or from the specified document (or document variant). The other two parameters, `inLastVersion` and `inLiveVersion`, are interpreted as booleans: 0 is false, any other (numeric) value is true.

If `inLastVersion` is true, only documents whose last version link to the specified document are included.

If `inLiveVersion` is true, only documents whose live version link to the specified document are included.

If both parameters are true or both are false, all documents are returned for which either the last or live version link to the specified document.

The optional parameter linktypes is a string containing a comma or whitespace separated list of the types of links to include, which is one or more of: `inline`, `out_of_line`, `image`, `include` or `other`.

4.6.2.10.3 IsLinked, IsNotLinked

```
IsLinked()
IsNotLinked()
```

`IsLinked()` evaluates to true for any document which is linked by other documents, `IsNotLinked()` evaluates to true for any document that is not linked from any other document (thus not reachable by following links in documents, the navigation tree, or linked by the content of other parts on which link extraction is performed).

4.6.2.10.4 HasPart

```
HasPart('partTypeName')
```

Searches documents which have a part of the specified part type. This search is version-dependent.

4.6.2.10.5 HasPartWithMimeType

```
HasPartWithMimeType('some mimetype')
```

Searches documents having a part with the given mime type. This search is version-dependent. This uses a 'like' condition, thus the % wildcard can be used in the parameter. For example, to search all images: `HasPartWithMimeType('image/%')`

4.6.2.10.6 DoesNotHaveVariant

```
DoestNotHaveVariant(branch, language)
```

Searches documents that do not have the specified variant. See also the page on variants (page 57) for more information.

4.6.2.11 Functions

The following functions can be used in value expressions.

4.6.2.11.1 String functions

4.6.2.11.1.1 Concat

Syntax:

```
Concat(value1, value2, ...) : string
```

Concatenates multiple strings. If the arguments are not strings, they are converted to a string using the same logic as the String function.

4.6.2.11.1.2 Length

Syntax:

```
Length(string) : long
```

Returns the length of its string argument.

4.6.2.11.1.3 Left

Syntax:

```
Left(string, length) : string
```

Returns 'length' leftmost characters from the string. If 'length' is larger than the string, the whole string is returned. If length is 0, an empty string is returned.

4.6.2.11.1.4 Right

Syntax:

```
Right(string, length) : string
```

Returns 'length' rightmost characters from the string. If 'length' is larger than the string, the whole string is returned. If length is 0, an empty string is returned.

4.6.2.11.1.5 Substring

Syntax:

```
Substring(string, position, length) : string
```

Returns a string formed by taking 'length' characters from the string at the specified position. The 'length' argument is optional, if not specified, it will go till the end of the input string. The 'position' argument starts at 1 for the first character.

4.6.2.11.1.6 UpperCase

Syntax:

```
UpperCase(string) : string
```

4.6.2.11.1.7 LowerCase

Syntax:

```
LowerCase(string) : string
```

4.6.2.11.1.8 String

Syntax:

```
String(value) : string
```

Converts its argument to a string.

Some of the behaviours:

- date and datetime values are formatted using the syntax for literals
- link values are formatted as "daisy:" links
- hierarchical values are formatted with slashes between the elements of the hierarchical path (e.g. "/A/B/C")
- multivalue values are formatted like this: [A,B,C]

4.6.2.11.2 Date and datetime functions

4.6.2.11.2.1 CurrentDate

Syntax:

```
CurrentDate(spec?) : date
```

Returns the current date.

The optional spec argument allows to specify an offset to the current date. It is a string with the following syntax:

```
+/- <num> (days|weeks|months|years)
```

For example:

```
CurrentDate('- 7 days')
```

4.6.2.11.2.2 CurrentDateTime

Syntax:

```
CurrentDateTime(spec?) : date
```

Returns the current datetime.

The optional spec argument allows to specify an offset to the current datetime. It is a string with the following syntax:

```
+/- <num> (seconds|minutes|hours|days|weeks|months|years)
```

For example:

```
CurrentDateTime('- 3 hours')
```

4.6.2.11.2.3 Year, Month, Week, DayOfWeek, DayOfMonth, DayOfYear

These functions all take a date or datetime as argument, and return a long value.

DayOfWeek returns a value in the range 1-7, where 1 is sunday.

For the Week function, the first week of the year is the first week containing a sunday.

4.6.2.11.2.4 RelativeDate, RelativeDateTime

These functions take one string argument consisting of 3 words, each one taken from the following groups:

```
start    this     week
end      last     month
         next     year
```

So for example:

```
RelativeDate('start this month')
```

returns a date set to the first day of the current month.

4.6.2.11.3 Numeric functions

4.6.2.11.3.1 +, -, * and /

The basic mathematical operations.

4.6.2.11.3.2 Random

Returns a pseudo-random double value greater than or equal to 0 and less than or equal to 1.

4.6.2.11.3.3 Mod

Syntax:

```
Mod(number1, number2)
```

4.6.2.11.3.4 Abs, Floor, Ceiling

These functions all take one number as argument.

4.6.2.11.3.5 Round

Syntax:

```
Round(number, scale)
```

Rounds the given number to have at most *scale* digits to the right of the decimal point.

4.6.2.11.4 Special

4.6.2.11.4.1 ContextDoc

Syntax:

```
ContextDoc(expression [, position])
```

In some cases a *context document* is available when performing a query. For example, when a query is embedded inside a document, that document serves as the context document. It is possible to evaluate expressions on this context document by use of this ContextDoc function. The optional position argument allows to climb up in the stack of context documents (which is available in publisher requests).

Examples:

```
ContextDoc(id) -- the id of the context document
ContextDoc($someField) -- the value of a field of the context document
ContextDoc(Concat(name, ' ', $someField))
```

4.6.2.11.4.2 UserId

Returns the ID of the current user (= the user executing the query).

```
UserId()  ->  function takes no arguments
```

4.6.2.11.4.3 Path

Converts its argument to a hierarchical path literal. This function is useful because there is no special query language syntax for entering hierarchical path literals. The argument should be a slash-separated hierarchical path, e.g.:

```
Path('/A/B/C')
Path('A/B/C')  -> the initial slash is optional
```

4.6.2.11.4.4 GetLinkPath

Syntax:

```
GetLinkPath(linkFieldName, includeCurrent, linkExpr)
```

Returns a hierarchical path formed by following a chain of documents linked through the specified link field.

The optional boolean argument includeCurrent indicates whether the current document should be part of the hierarchical path.

The optional linkExpr argument can be used to specify the start document, if it is not the current document.

Note that this expression can only be used in the select part of queries, or in the where clause if it is evaluated before performing the search. For example, as argument of matchesPath.

4.6.2.11.4.5 ReversePath

Reverses the order of the elements in a hierarchical path. For example useful in combination with GetLinkPath.

Syntax:

```
ReversePath(hierarchical-path)
```

4.6.2.12 Full text queries

4.6.2.12.1 FullText() function

For full text queries, the `where` part takes a special form. There are two possibilities: either only a full text search is performed, or the fulltext query is further restricted using 'normal' conditions. The two possible forms are:

```
... where FullText('word')
or
... where FullText('word') AND <other conditions>
for example:
... where FullText('word') AND $myfield = 'abc' AND InCollection('mycollection')
```

Note that the combining operator between the FullText condition and other conditions is always `AND`, thus the result of the full text query is further refined. The further conditions can of course be of any complexity, and can thus again contain `OR`.

The FullText clause needs to be the first after the word "where", it cannot appear at arbitrary positions in the where-clause.

If no order by clause is included when doing a full text query, the results are ordered according to the score assigned by the fulltext search engine.

The parameter of the `FullText(...)` function is a query which is passed on to the full text engine, in our case Lucene. See here[4].

The `FullText()` function can have 3 additional parameters which indicate if the search should be performed on the document name, document content or field content. By default, all three are searched. These parameters should be numeric: 0 indicates false, and any other value true.

For example:

```
FullText('word', 1, 0, 0)
```

Searches for 'word', but only in the document name.

Additionally, you can specify a branch and language as parameters to the `FullText` function, to specify that only documents of that branch/language should be searched. Thus the full syntax of the `FullText` function is:

```
FullText(lucene query, searchInName, searchInContent, searchInFields, branch, language)
```

Specifying the branch and language as part of the `FullText` function is more more efficient then using:

```
FullText(lucene query) and branch = 'my_branch' and language = 'my_language'
```

4.6.2.12.2 FullTextFragment() function

If you wish to have contextualized text fragments of the sought after terms. This function should be used in the select part of the query. By default this function will only return the first text fragment found. The fragments are returned as xml which has the following structure :

```
<html>
  <body>
    <div class="fulltext-fragment">
      ...  full text fragment ... <span class="fulltext-hit">the term</span> ... more
text ...
    </div>
    ...
  </body>
</html>
```

Usage of the function when you only wish to receive one fragment (default) :

```
select FullTextFragment() where FullText('word')
```

If you wish to have more text fragments you can specify the amount of fragments as a function parameter.

```
select FullTextFragment(5) where FullText('word')
```

This function will only return fragments from the content of the document. This means that context from document name or fields will not appear in the result.

4.6.2.13 The order by part

The `order by` part is optional.

The `order by` part contains a comma separated listing of value expressions, each of these optionally followed by `ASC` or `DESC` to indicate ascending (the default) or descending order. The expressions listed here have no connection with those in the `select`-part, i.e. it does not have to be subset of those.

"null" values are put at the end (when using ASC order).

4.6.2.14 The limit part

This can be used to limit the number of results returned from a query. This part is optional.

4.6.2.15 The option part

The `option` part allows to specify options that influence the execution of the query. The options are defined as:

```
option_name = 'option_value' (, option_name = 'option_value')*
```

Supported options:

name	value	default
include_retired	true/false	false
search_last_version	true/false	false
style_hint	(anything)	(empty)
annotate_link_fields	true/false	true
chunk_offset	an integer (start-index is 1)	N/A
chunk_length	an integer	N/A

include_retired is used to indicate that retired documents should be included in the result (by default they are not).

search_last_version is used to indicate that the last version of metadata should be searched and retrieved, instead of the live version. When using this, documents that do not have a live version will also be included in the query result (otherwise they are not included). Full text searches are always performed on the live data, regardless of whether this option is specified.

style_hint is used to supply a hint to the publishing layer for how the result of the query should be styled. The repository server does not do anything more then add the value of this option as an attribute on the generated XML query results (`<searchResult styleHint="my hint" ...`). It is then up to the publishing layer to pick this up and do something useful with it. For how this is handled in the DaisyWiki, see the page on Query Styling (page 196).

annotate_link_fields indicates whether selected fields of type "link" should be annotated with the document name of the document pointed to by the link. If you don't need this, you can disable this to gain some performance.

chunk_offset and **chunk_length** allow to retrieve a subset of the query results. This is useful for paged display of the query results.

4.6.3 Example queries

4.6.3.1 List of all documents

```
select id, name where true
```

4.6.3.2 Search on document name

```
select id, name where name like 'p%' order by creationTime desc limit 10
```

4.6.3.3 Show the 10 largest documents

```
select id, name, totalSizeOfParts where true order by totalSizeOfParts desc limit 10
```

4.6.3.4 Show documents of which the last version has not yet been published

```
select id, name, versionState, versionCreationTime
  where versionState = 'draft' option search_last_version = 'true'
```

4.6.3.5 Overview of all locks

```
select id, name, lockType, lockOwnerName, lockTimeAcquired, lockDuration
  where lockType is not null
```

4.6.3.6 All documents having a part containing an image

```
select id, name where HasPartWithMimeType('image/%')
```

4.6.3.7 Order documents randomly

```
select name where true order by Random()
```

4.6.3.8 Documents ordered by length of their name

```
select name, Length(name) where true order by Length(name) DESC
```

4.7 Full Text Indexer

Full text indexing in Daisy happens automatically when document variants are updated, so you do not need to worry about updating the index yourself. Technically, the full text indexer has a durable subscription on the JMS events generated by the repository, and it are these events which trigger the index updating.

4.7.1 Technology

Daisy uses Jakarta Lucene[5] as full-text indexer.

4.7.2 Included content

Only document variants which have a live version are included in the full text index. Thus retired document variants or document variants having only draft versions are not included. It is the content of the live version which is indexed, thus full text search operations always search on the live content.

For each document variant, the included content consists of the document name, the value of string fields, and text extracted from the parts. For the parts, text extraction will be performed on the data if the mime type is one of the following:

Mime type	Comment
text/plain	
text/xml	e.g. the "Daisy HTML" parts
application/xhtml+xml	XHTML documents
application/pdf	PDF files
application/vnd.sun.xml.writer	OpenOffice Writer files
application/msword application/vnd.ms-word	Microsoft Word files
application/mspowerpoint application/vnd.ms-powerpoint	Microsoft Powerpoint files
application/msexcel application/vnd.ms-excel	Microsoft Excel files

Support for other formats can be added by implementing a simple interface. Ask on the Daisy Mailing List if you need more information about this.

4.7.3 Index management

4.7.3.1 Optimizing the index

If you have a lot of documents in the repository, you can speed up fulltext searches by optimizing the index.

This is done as follows.

First go to the JMX console as explained in the JMX console documentation (page 323).

Follow the link titled: `Daisy:name=FullTextIndexer`

Look for the operation named `optimizeIndex` and invoke it (by pressing the Invoke button).

Afterwards, choose "Return to MBean view". In the `IndexerStatus` field, you will see an indication that the optimizing of the index is in progress. If you have a very small index, the optimizing might go so fast that it is already finished by the time you get back to that page. On larger indexes, the optimize procedure can take quite a bit of time.

4.7.3.2 Rebuilding the fulltext index

Rebuilding the fulltext index can be useful in a variety of situations:

- new or updated text extractors are available, and so you want to reindex the documents
- something went seriously wrong or you restored a backup and want to be sure the index is complete
- sometimes installing new Daisy versions requires rebuilding the index

The index can be rebuild for all documents or a selection of the documents.

If you want to completely rebuild the index, you might fist want to delete all the index files, which can be found in:

```
<daisydata dir>/indexstore
```

It is harmless to delete these, as the index can be rebuild at any time. Better don't delete them while the repository server is running though.

To trigger the rebuilding, go to the JMX console as explained in the JMX console documentation (page 323).

Follow the link titled: `Daisy:name=FullTextIndexUpdater`

In case you want to rebuild the complete index, invoke the operation named `reIndexAllDocuments`.

In case you want to rebuild the index only for some documents, you can use the operation `reIndexDocuments`. As parameter, you need to enter a query to select the documents to reindex. For example, to re-index all documents containing PDFs, you can use:

```
select id where HasPartWithMimeType('application/pdf')
```

What you put in the select-clause of the query doesn't matter.

After invoking the reindex operation, choose "Return to MBean view". Look at the attribute `ReindexStatus`. This will show the progress of the reindexing (refresh the page to see its value being updated). Or more correctly, of scheduling the reindexing. It is important that this ends completely before the repository server is stopped, otherwise the reindexing will not happen completely.

If you have a large repository, the `ReindexStatus` might show a long time the message "Querying the repository to retrieve the list of documents to re-index". This is because after just starting the repository, the documents still need to be loaded into the cache.

Note that the reindexing here only pushes reindex-jobs to the work queue of the fulltext indexer, the reindexing doesn't happen immediately.

To follow up the status of the actual indexing, go again to the start page of the JMX console, by choosing the "Server view" tab.

Over there, follow this link:
`org.apache.activemq:BrokerName=DaisyJMS,Type=Queue,Destination=fullTextIndexerJobs`

Look for the attribute named `QueueSize`. This indicates the amount of jobs waiting for the fulltext indexer to process. Each time you refresh this page, you will see this number go lower (or higher of new jobs are being added faster than they are processed).

If you have a large index, it could be beneficial to optimize it after the reindexing finished, as explained above.

4.8 User Management

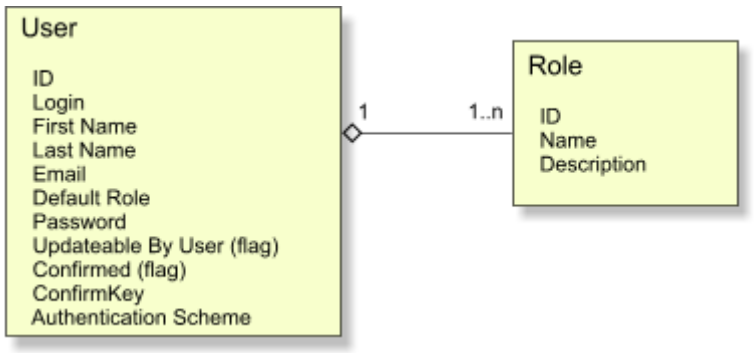
All operations done on the Daisy Repository Server are done as a certain user acting in a certain role(s). For this purpose, the Repository Server has a user management module to define the users and the roles. The authentication of the users is done by a separate component, allowing to plug in custom authentication techniques.

4.8.1 User Management

Users and roles are uniquely and permanently identified by a numeric ID, but they also have respectively a unique login and unique name.

A user has one or more roles. After logging in, it is both possible to have just one role active and let the user manually switch between his/her roles, or to have all roles of a user active at the same time (which is the behaviour traditionally associated with user groups). If a user has a default role, this role will be active after login. If no default role for the user is specified, all its roles will become active after login, with the exception of the Administrator role (if the user would have this role). This is because the Administrator role allows to do everything, which would then defeat the purpose of having other roles. If the user only has the Administrator role, then obviously that one will become active after login.

Users have a boolean flag called `updateable by user`: this indicates whether a user can update his/her own record. If true, a user can change its first name, last name, email and password. Role membership can of course not be changed, and neither can the login. It is useful to set this off for "shared users", for example the guest user in the Daisy Wiki application.



The `Confirmed` and `ConfirmKey` fields are used to support the well-known email-based verification mechanism in case of self-registration. If the `Confirmed` flag is false a user will not be able to log in.

4.8.1.1 The Administrator role

The repository server has one predefined role: *Administrator* (ID: 1). People having the role of Administrator as active role have a whole bunch of special privileges:

- they can access all documents in the repository and perform any operation on them. Thus the access control system doesn't apply to them.
- they can change the repository schema, manage users, manage collections, and manage the access control configuration.

4.8.1.2 Predefined users and roles

4.8.1.2.1 $system

$system is a bootstrap user internally needed in the repository. The user $system cannot log in, so its password is irrelevant. This user should not (and cannot) be deleted, nor should it be renamed. Simply don't worry about it.

4.8.1.2.2 internal

The user "internal" is a user created during the initialisation of the Daisy repository. The user is used by various components that run inside the repository server to talk to the repository. By default, we also use this user in the repository client component that runs inside Cocoon, which needs a user to update its caches.

The internal user has (and should have) the Administrator role.

During installation, this user gets assigned a long random generated password (you can see it in the myconfig.xml or cocoon.xconf).

4.8.1.2.3 guest user and guest role (Daisy Wiki)

The Daisy Wiki predefines a user called guest and a role called guest. This user has the password "guest". This is the user that becomes automatically active when surfing to the Daisy Wiki application, without needing to log in. After initialisation of the Daisy Wiki, the ACL is configured to disallow any write operations for users having the guest role.

4.8.1.2.4 registrar (Daisy Wiki)

The registrar user is the user that will:

- create and update user accounts during the self-registration
- reset passwords and do email-based lookup of logins in case of forgotten passwords or logins

During installation, this user gets assigned a long random generated password (you can see it in the cocoon.xconf).

4.8.2 Authentication Schemes

Daisy provides its own password authentication, but it is also possible to delegate the authentication to an external system. At the time of this writing, Daisy ships with support for authentication using LDAP and NTLM. It is possible to configure multiple *authentication schemes* and to have different users authenticated against different authentication schemes.

The authentication schemes are configured in the `myconfig.xml` file (which is located in `<daisy-data-dir>/conf`). Just search on "ldap" or "ntlm" and you'll see the appropriate sections. After making changes there, you will need to restart the repository server. To let users use the newly defined authentication scheme(s), you need to edit their settings via the user editor on the administration pages.

Daisy does not do automatic synchronisation of user information (such as updating the e-mail address based on what is stored in LDAP), but it is possible to auto-create users on first log in. This means that when a user logs in for the first time in Daisy, and does not yet exist in Daisy, an authentication scheme is given the possibility to create the user (if it exist in the external system). To enable this feature, search in the `myconfig.xml` file for "authenticationSchemeForUserCreation".

To debug authentication problems, look at the log files in <daisy-data-dir> /logs/daisy-request-errors-<date>.log. Problems in the configuration of the authentication schemes do not ripple through over the HTTP interface of the repository, thus are not visible in the Daisy Wiki.

4.8.2.1 Implementing new authentication schemes

For a tutorial, see here (page 137).

For real samples, simply look at the source code of the NTLM and LDAP schemes. For this, download the Daisy source code, you'll find them in the following directories:

```
services/ldap-auth
services/ntlm-auth
```

4.9 Access Control

4.9.1 Introduction

This document explains Daisy's features for access control: the authorisation of document operations such as read and write.

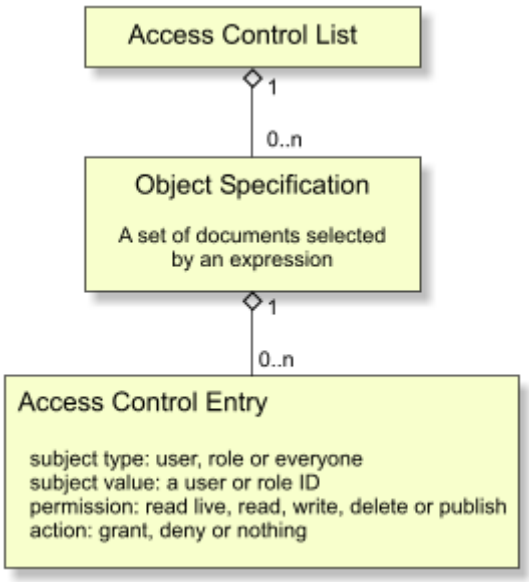
While we usually talk about documents, technically the access control happens on the document variant level: a user is granted or denied access to a certain document variant.

In many systems, access control is configured by having access control lists (ACLs) attached to documents. These ACLs contain access control rules which tell for a certain users or roles (groups) what operations they can or cannot perform.

For Daisy, it was considered to be too laborious to manage ACLs for each individual document. Therefore, there is **one global ACL**, where you can select sets of documents based on an expression and then define the access control rules that apply to these documents.

4.9.2 Structure of the ACL

The structure of the ACL is illustrated by the diagram below.



In ACL terminology, an *object* is the protected resource, and a *subject* is an entity wanting to perform an operation on the object. The objects in our case are documents, selected using an

expression. The subjects are users, which can be living organisms, usually humans, or programs acting on behalves of them.

As will become clear when reading about the evaluation of the ACL below, the order of the entries in the ACL is important.

4.9.2.1 Object specification

The expression used to select documents in the object specification uses the same syntax as in the `where` clause of an expression in the Daisy Query Language (page 63). However, the number of identifiers that are available is severely limited. More specifically, you can test on the following things:

- the document type
- collection membership (using the `InCollection` function)
- document ID (to have rules specific to one document)
- fields for which the ACL-allowed flag of the field type is set to true
- the branch and language

Some examples of expressions:

```
InCollection('mycollection')

documentType = 'Navigation' and InCollection('mycollection')

$myfield = 'x' or $myotherfield = 'y'
```

For the evaluation of these expression, the data of the fields in the last version is used, *not* the data from the live version.

4.9.2.2 Access Control Entry

See diagram.

If the subject type is everyone, the subject value should be set to -1.

If you give 'read live' rights to someone, they are able to:

- read 'live data' of documents, this means: all non-versioned data, and the data from the live version.
  Access to retired documents is denied.
  Getting the list of versions of the document is not allowed.
  In query results, documents without a live version will not appear (if the option `search_last_version` is specified, documents only appear if the last version is the live version)
- add comments to documents

If you give 'read' rights to someone, they have full read access to the document (thus they can view all versions and the list of versions).

If you give 'write' rights to someone, they are able to:

- create documents
- update (save) documents
- take a lock on the documents

The 'delete' right gives users the possibility to delete documents or document variants.

If you give 'publish' rights to someone, they are able to change the publish/draft state of versions of documents.

4.9.2.3 Staging and Live ACL

In Daisy, there are two ACLs: a staging ACL and a live ACL. Only the staging ACL is directly editable. The only way to update the ACL is to first edit the staging ACL, and then put it live (= copy the staging ACL over the live ACL).

Before putting it live, it is possible to first test the staging ACL: you can give a document id, a role id and a user id and get the result of ACL evaluation in return, including an explanation of which ACL rules made the final decision.

In the Daisy Wiki front end, all these operations are available from the administration console. It is recommended that after editing the ACL, you first test it before putting it live, so that you are sure there are no syntax errors in the document selection expressions.

4.9.3 Evaluation of the ACL: how is determined if someone gets access to a document

The determination of the authorisation of the various operations for a certain document happens as follows:

1. If the user is acting in the role of Administrator, the user has full access rights. The ACL is not checked.
2. If the user is owner of the document, the user always has read, write and delete rights. Publish rights are still determined by the ACL.
3. If the document is marked as private and the user is not the owner of the document, all rights are denied. The ACL is not checked.
4. The ACL result is initialised to deny all access (read live, read, write, publish and delete), and the ACL is evaluated from top to bottom:

    - If an object expression evaluates to true for the document, the access control entries belonging to that object specification are checked
    - If the subject type and subject value of an access control entry matches, the permissions defined in that entry override any previous result
    - The evaluation of the ACL does not stop at the first matching object or subject, but goes further till the bottom.

6. At the end of the ACL evaluation some further checks are performed:

- if the user does not have 'read live' rights, any other rights are denied too
- if the user does not have read rights, the write and publish rights are denied too
- if the user does not have write rights, the delete right is denied too.

Further notes:

- when saving a document, the ACL is always checked on the document currently stored, not on the newly edited document (unless it is a new document). This is because the ACL evaluation result can depend on the value of fields, and the user might have edited those fields to try to gain access to the document.
- A user cannot change a document in such a way that the user itself has no write rights anymore to the document, e.g. by changing collection membership or field values.
- The ACL is only concerned with authorisation of rights on documents. Other permissions, like who can manage users, change the ACL, create document types, etc... is simply managed via the Administrator role: users acting in the Administrator role can do all those, others can't.

4.9.4 Other security aspects

This document only discussed authorisation of operations on documents for legitimate users. Other aspects of security include:

- authentication: see User Management (page 82)
- audit logging: since Daisy generates JMS events for all (write) operations happening on the repository, you could get a full audit log by logging all these events. The content of these events are XML descriptions of the changes (usually an XML dump of the entity before and after modification)
- physical protection of the data: if someone can access the filesystem on which the parts are stored, or the relational database, they can see and/or modify anything
- integrity: hasn't anyone been altering the data before delivery to the user. Here the use of https can help.

4.10 Email Notifier

4.10.1 General

Daisy can send out emails when changes are made to documents. To make use of this the SMTP host must be correctly configured, which is usually done as part of the installation, but can be changed afterwards (see below). In the Daisy Wiki, individual users can subscribe to get notifications by selecting the "User Settings" link, making sure their email address is filled in, and checking the checkbox next to "Receive email notifications of document-related changes.".

Users will only receive events of documents to which they have at least read (not 'read live') access rights. It is possible to receive notifications for individual documents, for all documents

belonging to a certain collection, or for all documents. The mails will notify document creation, document updates or version state changes.

While we usually talk about documents, the actual notifications happen on the document variant level.

As you can see on the User Settings page, it is also possible to subscribe to other events: user, schema, collection and ACL related changes. However, for these events proper formatting of the mails is not yet implemented, they simply contain an XML dump of the event.

4.10.2 Configuration

Configuration of the email options happens in the `<DAISY_DATA>/conf/myconfig.xml` file. There you can configure:

- the SMTP server
- the from address for the emails
- the URLs for documents, so that the URL of the changed document can be included in the emails
- users who's events should be ingored (see below)
- whether email notifications should be enabled (a global enable/disable flag)

After making any changes to the `myconfig.xml` file, the repository server needs to be restarted.

4.10.3 Ignoring events from users

Sometimes when doing automated document updates, a lot of change events might be produced, and it can be undesirable to produce email notifications for these events.

For this, the email notifier can be configured to ignore the events caused by certain users. For example, if you have an application which connects as user "john" to the repository server, then it is possible to say that events (document updates etc.) caused by john should not result in email notifications.

The users who's events should be ignored can be configured in the myconfig.xml (use this for permanent settings), but can also be changed at runtime through the JMX console (page 323) (use this for temporarily disabling notifications for a user).

4.10.4 Implementation notes

The email notifier is an extension component running inside the repository server. It is independent of the Daisy Wiki. The email notifier provides a Java API for managing the subscriptions, as well as additions to the HTTP+XML interface (logical, because that's how the implementation of the Java API talks to the repository).

4.11 Document Task Manager

The purpose of the Document Task Manager (DTM) is to perform a certain task across a set of documents. The DTM is an optional component running inside the Daisy Repository Server. Some of its features are:

Tasks are **executed in the background**, inside the repository server. Thus the user (a person or another application) starting the task does not have to wait until it is completed, but can do something else and check later if the task ended successfully.

The **execution progress** of the task is **maintained persistently** in the database. For each document on which the task needs to be executed, you can consult whether it has been performed successfully, whether it failed (and why), or whether it still has to be executed. Since this information is tracked persistently in the database, it is not lost in case the server would be interrupted.

Tasks can be interrupted. Since the task is performed on one document after another, it is easily possible to interrupt between two documents.

Tasks can be **written in Javascript or be composed from built-in actions**. Executing custom Javascript-based tasks is only allowed by Administrators, since there is a certain risk associated with it. For example, it is possible to write a task containing an endless loop which would only be interruptible by shutting down the repository server, or a task could call `System.exit()` to shut down the server.

The execution details of a task, which are stored in the database, are cleaned up automatically after two weeks (by default), and can of course also be deleted manually.

The DTM is accessible via the HTTP API and the Java API.

The Daisy Wiki contains a frontend for starting new tasks and consulting the execution details of existing tasks.

Ideas for the future:

- scheduled execution of document tasks
- adding more built-in actions (the ones currently available are mainly to support working with document variants)

4.12 Publisher

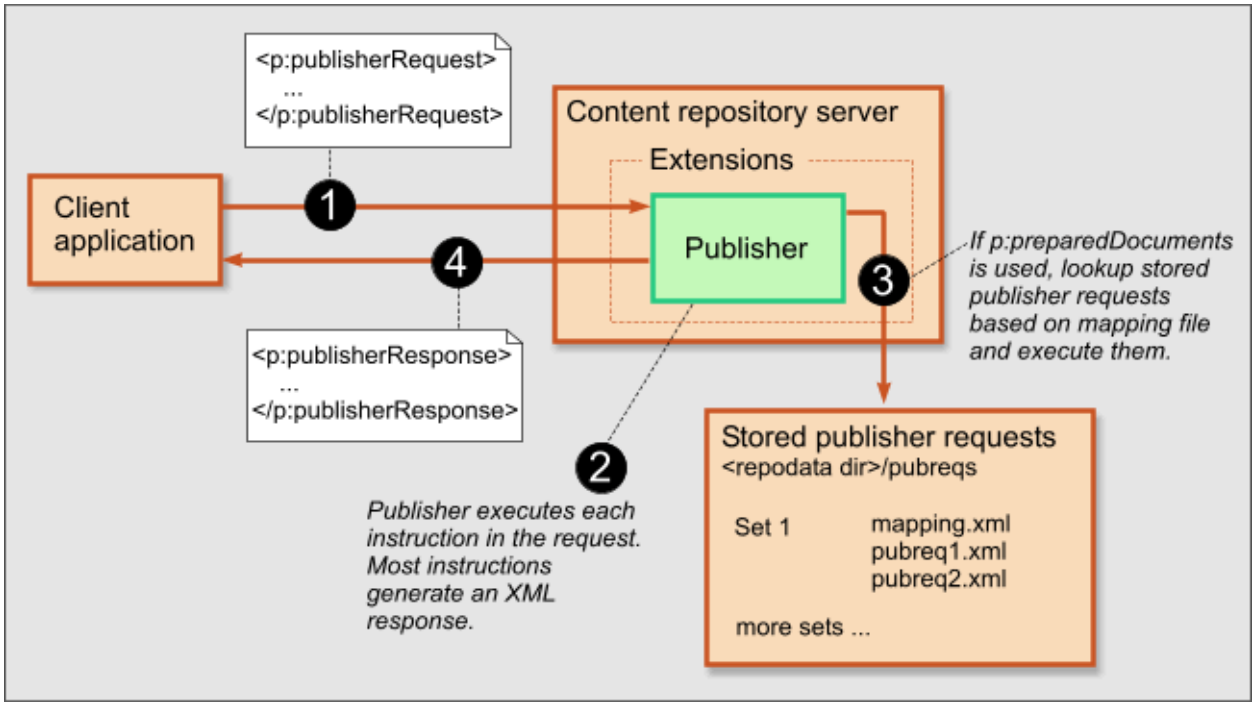
4.12.1 Introduction

The publisher is an extension component running in the content repository server.

Its original goal was to retrieve in one remote call the information you need to display on a page. The result is returned as an XML document.

However, the information that can be requested from the publisher consists of more than just XML dumps of repository entities. The publisher provides all sorts of extra functionality, such as 'prepared documents' for publishing with support for document-dependent content aggregation, or performing diffs between document versions.

The publisher was developed to support the needs of the Daisy Wiki, but is useful for other applications as well. Suggestions for features (or patches) are of course welcomed.



4.12.2 The publisher request format

A publisher request is an XML document with as root element p:publisherRequest (page 106), and containing various instructions. This is the full list of available instructions:

Name
p:aclInfo
p:annotatedDocument
p:annotatedVersionList
p:availableVariants
p:choose
p:comments
p:diff
p:document
p:forEach
p:group
p:ids
p:if
p:myComments
p:navigationTree
p:performFacetedQuery
p:performQuery

p:preparedDocuments (& p:prepareDocument)
p:publisherRequest
p:resolveDocumentIds
p:resolveVariables
p:selectionList
p:shallowAnnotatedVersion
p:subscriptionInfo
p:variablesConfig
p:variablesList

4.12.3 Concepts

4.12.3.1 Two kinds of publisher requests

A publisher request takes the form of an XML document, describing the various stuff you want the publisher to return. The publisher request is send to the Publisher component, and the Publisher answers with a big XML response.

Next to the publisher requests that are sent to the Publisher, the Publisher can also execute additional publisher requests as part of the p:preparedDocuments (page 103) instruction. These additional publisher requests are stored in a directory accessible by the Publisher, usually this is:

```
<repodata dir>/pubreqs/
```

The format of these publisher requests is exactly the same.

4.12.3.2 Context document stack

The p:document (page 96) instruction in a a publisher request pushes a document on the context document stack. A good part of the publisher instructions need a context document based on which they will work.

In expressions or in queries (such as in p:performQuery (page 103)), the context document stack can be accessed using the ContextDoc(expr[, level]) function. The optional level argument of the ContextDoc function describes how high to go up in the context doc stack. See the query language reference (page 63) for details.

4.12.3.3 Expressions

The parameters of some publisher instructions, specified in attributes and child-elements, can contain expressions, rather than just a literal value.

To specify an expression, the attribute or element must start with `${` and end on `}`. For example:

```
<element attribute="${some expr}">${some expr}</element>
```

Using multiple expressions or having additional content around the expression is not supported.

The expressions are Daisy query-language expressions. The identifiers apply to the current context document. For example the expression `${id}` would evaluate to the ID of the current context document.

4.12.4 Testing a publisher request

The Publisher can be easily called using the HTTP interface (page 119). Just create an XML file containing the publisher request, and submit it using a tool like wget, which is available on many Unix systems (there's a Windows version too).

For example, create a file called pubreq.xml containing something like this:

```
<?xml version="1.0"?>
<p:publisherRequest
  xmlns:p="http://outerx.org/daisy/1.0#publisher"
  locale="en-US">

  <p:document id="1-DSY">
    <p:aclInfo/>
    <p:availableVariants/>
    <p:annotatedDocument/>
    <p:annotatedVersionList/>
  </p:document>

</p:publisherRequest>
```

The above example assumes a document with id 1-DSY exists. If not, just change the document id.

Now we can execute the publisher request:

```
wget --post-file=pubreq.xml --http-user=testuser@1
     --http-passwd=testuser http://localhost:9263/publisher/request
```

See the HTTP API documentation (page 119) for more examples on using wget.

wget will save the response in a file, typically called request. You can open it in any text or XML editor, but to view it easily readable you can use:

```
xmllint --format request | less
```

4.12.5 About the instruction reference

In general, the reference of the publisher instructions only displays the request syntax, and not the format of the responses. Example responses can be easily obtained by executing a publisher request.

4.12.6 p:aclInfo

Returns the result of evaluating the current context document against the ACL (page 85).

4.12.6.1 Request

This request requires no attributes, so its syntax is simply:

```
<p:aclInfo/>
```

4.12.6.2 Response

The response is a d:aclResultInfo element.

4.12.7 p:annotatedDocument

Returns the XML representation of the current document with some annotations. The annotations include things like the name and label of the document type, display names for users, branch and language name (for all these things, otherwise only the numeric IDs would be present), and annotations to the fields.

4.12.7.1 Request

Syntax:

```
<p:annotatedDocument [inlineParts="..."] />
```

In contrast with most other elements, if you request the live version of the document but the document doesn't have a live version, this element will automatically fallback to the last version, so there will always be a d:document element in the response.

The p:annoatedDocument element can have an optional attribute called inlineParts, which can have as value "#all" or "#daisyHtml". In case #daisyHtml is specified, the part content of all Daisy-HTML parts will be inlined (though without any processing applied to them, thus unlike p:preparedDocuments (page 103)). When #all is specified, the content of all parts whose mime-type starts with "text/" will be inlined. If it is an XML mimetype, the content is inserted as XML. [in the future, this attribute could be expanded so that it takes a comma-separated list of part type names whose content needs to be inlined]

4.12.7.2 Response

A d:document element with annotations.

4.12.8 p:annotatedVersionList

Returns a list of all versions of the document, with some annotations on top of the default XML representation of a version list.

4.12.8.1 Request

Syntax:

```
<p:annotatedVersionList/>
```

4.12.8.2 Response

A d:versions element.

4.12.9 p:availableVariants

Returns the available variants for the current context document.

4.12.9.1 Request

Syntax:

```
<p:availableVariants/>
```

4.12.9.2 Response

A d:availableVariants element.

4.12.10 p:choose

Allows to execute one among of a number of possible alternatives.

4.12.10.1 Request

Syntax:

```
<p:choose>
  <p:when test="...">
     [... any publisher instruction ...]
  </p:when>

  [... more p:when's ...]

  <p:otherwise>
    [... any publisher instruction ...]
  </p:otherwise>
</p:choose>
```

There should be at least one p:when, the p:otherwise is optional.

The test attributes contains an expression as in the where-clause of the query language (page 63). Since it always contains an expression, no ${...} should be used.

4.12.10.2 Response

p:choose itself doesn't generate any response, only the response from the executed alternative will be present in the output.

4.12.11 p:comments

Returns the comments for the current context document.

4.12.11.1 Request

Syntax:

```
<p:comments/>
```

4.12.11.2 Response

A d:comments element. The newlines in the comments will be replaced with
 tags.

4.12.12 p:diff

Returns a diff of the current context document/version with another version of this document or another document.

4.12.12.1 Request

Syntax:

```
<p:diff contentDiffType="text|html|htmlsource">
  <p:otherDocument id="expr"  branch="expr" language="expr" version="expr"/>
</p:diff>
```

If no p:otherDocument element is specified, the diff will automatically be taken with the previous version of the document. If there is no such version (because the document has only one version yet), there will be no diff response.

If a p:otherDocument element is supplied, any combination of attributes is allowed, all attributes are optional.

The contentDiffType attribute is optional, text is the default. Specify 'html' for a visual HTML compare, 'htmlsource' does in an inline HTML source diff (rather than a line-based diff). This attribute only has effect on Daisy-HTML parts.

4.12.12.2 Response

A diff-report element, except in the case no output is generated because the from or to version is not available.

4.12.13 p:document

A p:document request is push a document on the context document stack, and thus to change the currently active context document. The context document is the document on which the document related requests apply.

4.12.13.1 Request

Any publisher request element can be nested within p:document.

The p:document request can work in three ways

4.12.13.1.1 (1) Specify the document explicitly

Syntax:

```
<p:document id="..." branch="..." language="..." version="...">
  [ ... child instructions ... ]
</p:document>
```

Using the attributes id, branch (optional), language (optional) and version (optional) the new context document is specified.

The branch and language can be specified either by name or ID. If not specified, they default to the main branch and default language.

The version can be specified as "live" (the default), "last" or an explicit version number.

4.12.13.1.2 (2) Specify a field attribute

Syntax:

```
<p:document field="..." hierarchyElement="...">
  [ ... child instructions ... ]
</p:document>
```

When the p:document is used in a location where there is already a context document (e.g. from a parent p:document), it is possible to use an attribute called field. The value of this attribute should be the name of a link-type field. This p:document request will then change the context document to the document specified in that link-type field in the current context document. If the current context document does not have such a field, the p:document request will be silently skipped. If the link-type field is a multivalue field, the p:document request will be executed once for each value of the multivalue field. This will then lead to multiple sibling p:document elements in the publisher response.

Exactly the same can also be achieved through the p:forEach (page 97) request. In fact, the internal implementation uses p:forEach so this is just an alternative (older) syntax for the same thing.

See also p:forEach for an explication of the hierarchyElement attribute.

4.12.13.1.3 (3) Implicit

Sometimes the p:document instruction is used as the child of other instructions such as p:forEach or p:navigationTree. In that case the context document is determined by the parent instruction, and should hence not be specified.

4.12.13.2 Response

The p:document request will output a p:document element in the publisherResponse. This element will have attributes describing the exact document variant and version that the context document was changed to.

4.12.14 p:forEach

Executes publisher instructions for each document in a list of documents. The list of documents on which to operate can either result from a query or an expression.

4.12.14.1 Request

4.12.14.1.1 Query

Syntax:

```
<p:forEach useLastVersion="true|false">
  <p:query>select ... where ... order by ...</p:query>
  <p:document>
    [ ... child instructions ... ]
  </p:document>
</p:forEach>
```

If the useLastVersion attribute is false or not specified, the live version of each document will be used, otherwise the last version.

If there is a context document available then the ContextDoc function can be used in the query or expression.

4.12.14.1.2 Expression

Syntax:

```
<p:forEach useLastVersion="true|false">
  <p:expression [precompile="true|false"] [hierarchyElement="all|an integer"]>...<
/p:expression>
  <p:document>
    [ ... child instructions ... ]
  </p:document>
</p:forEach>
```

The expression is an expression using the Daisy query language syntax, and should return link values.

The optional *precompile* attribute indicates whether the expression should be compiled just once or recompiled upon each execution. Usually one should leave this to its default true value.

When the value returned by the expression is a hierarchy path (or a multivalue of hierarchy paths), then p:forEach will by default run over all the values in the hierarchy path. The optional *hierarchyElement* attribute can be used to select just one element from the hierarchy path. This attribute accepts integer values and '*all*'. If you do not specify the hierarchy element attribute then 'all' will be used by default.

- *all* wil use all the elements in the path as context documents.
- A positive number will select the n^{th} element starting from the beginning of the path.
- A negative number will select the n^{th} element starting from the end of the path.

An illustration. Consider the following hierarchy: /domain/kingdom/subkingdom/branch/infrakingdom. Using 2 as hierarchyElement will get you documents at the level of kingdom. Using -2 will get you documents at the level of branch. As you can see the hierarchyElement specification is 1-based. Choosing level 12 will get you nothing since the hierarchy does not have 12 levels.

4.12.14.1.2.1 Examples

If the current context document has a link field (single or multi-value) called MyField, then you could run over its linked documents as follows:

```
<p:forEach useLastVersion="true|false">
  <p:expression>$MyField</p:expression>
  <p:document>
    <p:annotatedDocument/>
```

```
  </p:document>
</p:forEach>
```

If you have a number of documents which are linked in a chain using a certain field (e.g. Category documents with a link field pointing to their parent category), then the GetLinkPath function is a useful tool:

```
<p:expression>GetLinkPath('CategoryParent')</p:expression>
```

4.12.14.2 Response

Zero or more p:document elements.

4.12.15 p:group

The p:group element acts as a container for other instructions. It allows to distinguish between e.g. different queries or navigation tree results if you would have more than one of them.

4.12.15.1 Request

Syntax:

```
<p:group id="expr" catchErrors="true|false">

 [... child instructions ...]

</p:group>
```

The `id` attribute is required.

The `catchErrors` attribute is optional. When this attribute has the value true, any errors that occur during the processing of the children of the p:group element will be caught. The result will then be a p:group element with an attribute error="true" and as child the stacktrace of the error. When catchErrors is true, the result of the execution of the children of the p:group element will need to be buffered temporarily, so only use this when you really need it.

4.12.15.2 Response

A p:group element with id attribute, and contained in it the output of the child instructions.

4.12.16 p:ids

Returns the list of all values of the id attributes occurring in the Daisy-HTML parts of the current context document. This can be useful in editors to show the user a list of possible fragment identifier values.

4.12.16.1 Request

Syntax:

```
<p:ids/>
```

4.12.16.2 Response

```
<p:ids>
  [ zero or more child p:id elements ]
  <p:id>....</p:id>
</p:ids>
```

4.12.17 p:if

Allows to execute a part of the publisher request only if a certain test is satisfied.

4.12.17.1 Request

Syntax:

```
<p:if test="...">
  [... child instructions ...]
</p:if>
```

The test attribute specifies a conditional expression (an expression evaluating to true or false) in the same format as used in the Daisy query language.

For example:

```
<p:if test="$MyField > 20">
  ...
</p:if>
```

4.12.17.2 Response

If the test evaluated to true, the output of the child instructions, otherwise nothing.

4.12.18 p:myComments

Returns a list of all private comments of the user.

4.12.18.1 Request

Syntax:

```
<p:myComments/>
```

4.12.18.2 Response

Same as for p:comments (page 95).

4.12.19 p:navigationTree

Request a navigation tree (page 166) from the Navigation Manager.

4.12.19.1 Request

The full form of this request is:

```
<p:navigationTree>
  <p:navigationDocument id="expr" branch="expr" language="expr"/>
  <p:activeDocument id="expr" branch="expr" language="expr"/>
  <p:activePath>expr</p:activePath>
  <p:contextualized>true|false</p:contextualized>
  <p:depth>...</p:depth>
  <p:versionMode>live|last</p:versionMode>
  <p:document>...</p:document>
</p:navigationTree>
```

The elements p:activeDocument, p:activePath, p:versionMode, p:depth and p:document elements are optional.

To make the active document the current context document, one can use expressions like this:

```
<p:navigationTree>
  <p:navigationDocument id="338-DSY"/>
  <p:activeDocument id="${id}" branch="${branch}" language="${language}"/>
</p:navigationTree>
```

4.12.19.1.1 Attaching additional information to the document nodes

If a p:document (page 96) instruction is present, then the p:document will be executed for each document node in the navigation tree, and the response is inserted as first child of each element. This provides the ability to annotate the document nodes in the navigation tree with additional information of the document, so that you can e.g. know the document type of the document. One could even aggregate the full content of the documents, if desired.

An example:

```
<p:navigationTree>
  <p:navigationDocument id="338-DSY"/>
  <p:document>
    <p:annotatedDocument/>
  </p:document>
</p:navigationTree>
```

4.12.19.2 Response

The response of this instruction is a n:navigationTree element, with n denoting the navigation namespace.

4.12.20 p:performFacetedQuery

Returns the result of executing a query.

4.12.20.1 Request

Syntax:

```
  <p:performFacetedQuery>
    <p:options>
      <p:additionalSelects>
        <p:expression>name</p:expression>
```

```
        <p:expression>summary</p:expression>
      </p:additionalSelects>
      <p:defaultConditions>documentType='SimpleDocument'</p:defaultConditions>
      <p:defaultSortOrder>name asc</p:defaultSortOrder>
      <p:queryOptions>
        <p:queryOption name="include_retired" value="true"/>
      </p:queryOptions>
    </p:options>
    <p:facets>
      <p:facet expression="variantLastModifierLogin" maxValues="10" sortOnValue="true"
sortAscending="false" type="default">
        <p:properties>
          <p:property name="" value=""/>
        </p:properties>
      </p:facet>
    </p:facets>
  </p:performFacetedQuery>
```

This syntax is similar to the one for the faceted browser definition (page 173).

4.12.20.1.1 Options

All options are, as the name says, optional. The options element is not optional.

- **p:defaultConditions** expects a clause that would go in the where part of a Daisy query. If this element is missing then it will default to 'true'.
  If there is a context document available (i.e. if this p:performFacetedQuery is used inside a p:document) then the ContextDoc function can be used in the query.
- **p:defaultSortOrder** specifies the order in which the search results should be sorted.
- **p:queryOptions** is an element that sets query options like 'search_last_version'.
- **p:additionalSelects** allows specifications of identifiers other than those of the facets.

4.12.20.1.2 Facets

Each facet element defines a facet.
Attributes :

- **expression** is the only required part of a facet definition. It expects a query expression to build the facet with.
- **maxValues** sets the maximum amount of facet values to be returned. This attribute is optional and defaults to unlimited (-1).
- **sortOnValue** sets whether facets should be sorted according to their value(true) or count(false). This is optional and defaults to true.
- **sortAscending** sets whether facets should be sorted ascending(true) or decending(false). Defaults to true.
- **type** sets the facet type. These are the same types as the facet browser (page 176). Defaults to 'default'.

4.12.20.2 Response

The result of a query is a d:facetedQueryResult element.

4.12.21 p:performQuery

Returns the result of executing a query.

4.12.21.1 Request

Syntax:

```
<p:performQuery>
  <p:query>select ... where ... order by ...</p:query>

  [ optional elements: ]
  <p:extraConditions>...</p:extraConditions>
  <p:document>...</p:document>

</p:performQuery>
```

If there is a context document available (i.e. if this p:performQuery is used inside a p:document) then the ContextDoc function can be used in the query.

The optional p:extraConditions element specifies additional conditions that will be AND-ed with those in the where clause of the query. This feature is not often needed. It can be useful when you let the user enter a free query but want to enforce some condition, e.g. limit the documents to a certain collection.

The optional p:document element can contain publisher instructions that will be performed for each row in the query result set, their result of them will be inserted as child elements of the row elements. On each result set row, the document corresponding to the row is pushed as context document. This functionality is similar to what can be done with p:forEach (page 97), but has the advantage that information about the query such as chunk offset and size stays available.

4.12.21.2 Response

The result of a query is a d:searchResult element.

4.12.22 p:preparedDocuments (& p:prepareDocument)

p:preparedDocuments is the most powerful of all publisher requests. It returns the content of the document prepared for publishing. The preparation consists of all sorts of things such as:

- inlining the content of Daisy-HTML parts in the document XML.
- executing queries and query-includes embedded inside documents
- annotating images with the name of target document and the file name of the ImageData part.
- annotating "daisy:" links with the name of the target document and the path in the navigation tree (if navigation tree details have been supplied)
- processing includes

A very important point of p:preparedDocuments is that is able to use secondary publisher requests for the requested document and each included document. The publisher requests to use are determined based on a mapping file and allow to aggregate additional information withoccu the document based on e.g. its document type.

4.12.22.1 Request

Syntax:

```
<p:preparedDocuments publisherRequestSet="..."
                     displayContext="free string"
                     applyDocumentTypeStyling="true|false">
  <p:navigationDocument id="..." branch="..." language="..."/>
</p:preparedDocuments>
```

The `applyDocumentTypeStyling` and `displayContext` attributes are not used by the publisher, but are simply replicated in the result.

In the Daisy Wiki, their purpose are:

- applyDocumentTypeStyling: indicates if document styling should be automatically applied on preparedDocuments occurring in a publisher response (the "Type" in the attributes name is because historically the document styling is dependent on the type of document)
- displayContext: a freely choosen string indicating in which context the document is displayed. This string is made available to the document-styling XSLT. It allows to alter the styling depending on whether the document is displayed standalone or aggregated with some other document (or as part of a feed, or whatever).

The publisherRequestSet attribute: see below.

The p:navigationDocument element is optional. If supplied, it enables to annotate "daisy:" links with the path where they occur in the navigation tree.

4.12.22.1.1 How it works

p:preparedDocuments looks up a new publisher request to be performed on the context document. The publisher request to be used can be determined dynamically (described further on), but by default it is this one:

```
<p:publisherRequest xmlns:p="http://outerx.org/daisy/1.0#publisher">
  <p:prepareDocument/>
  <p:aclInfo/>
  <p:subscriptionInfo/>
</p:publisherRequest>
```

This publisher request should (usually) contain a <p:prepareDocument/> element. The p:prepareDocument will be replaced by the Daisy document as XML (d:document), in which the HTML content is inlined and processed (i.e. the things mentioned in the enumeration above). If the content contains an include of another document, then for this included document the publisher will again determine a publisher request to be performed upon it, and execute it. The same happens for each include (recursively). The results of all these publisher requests are inserted in the publisher response in a structure like this:

```
<p:preparedDocuments applyDocumentTypeSpecificStyling="true|false">

  <p:preparedDocument id="1">
    <p:publisherResponse>
      <d:document ...
    </p:publisherResponse>
  </p:preparedDocument>

  <p:preparedDocument id="2">
    <p:publisherResponse>
```

```
      <d:document ...
    </p:publisherResponse>
  </p:preparedDocument>

</p:preparedDocuments>
```

The publisher response of the context document will always end up in the p:preparedDocument element with attribute id="1". If the document includes no other documents, this will be the only p:preparedDocument. Otherwise, for each included document (directly or indirectly), an additional p:preparedDocument element will be present.

So the included documents are not returned in a nested structure, but as a flat list. This allows to perform custom styling on each separate document before nesting them.

On the actual location of an include, a p:daisyPreparedInclude element is inserted, with an id attribute referencing the related p:preparedDocument element.

The content of a p:preparedDocument element is thus a single p:publisherResponse element, which in turns contains a single d:document element (as result of the p:prepareDocument in the publisher request). This d:document element follows the standard form of XML as is otherwise retrieved via the HTTP interface or by using the getXml() method on a document object, but with lots of additions such as inlined content for Daisy-HTML parts and non-string attribute values formatted according to the specified locale.

If you requested the live version of the document, but the document does not have a live version, there will simply be no p:preparedDocuments element in the response.

4.12.22.1.2 Determination of the publisher request to be performed

If instead of the default publisher request mentioned above, you want to execute some custom publisher request (which can be used to retrieve information related to the document being published), then this is possible by defining a *publisher request set*.

In the data directory of the repository server, you will find a subdirectory called 'pubreqs'. In this directory, each subdirectory specifies a publisher request set. Each such subdirectory should contain a file called `mapping.xml` and one or more other files containing publisher requests.

The mapping.xml file looks like this:

```
<m:publisherMapping xmlns:m="http://outerx.org/daisy/1.0#publishermapping">
  <m:when test="documentType = 'mydoctype'" use="myrequest.xml"/>
  <m:when test="true" use="default.xml"/>
</m:publisherMapping>
```

The publisher will run over each of the when rules, and if the expression in the test attribute matches, it will use the publisher request specified in the use attribute. The expressions are the same as used in the query language, and thus also the same as used in the ACL definition.

To make use of such a specific set of publisher requests, you use the publisherRequestSet attribute on the p:preparedDocuments element. The value of this attribute should correspond to the name of subdirectory of the pubreqs directory.

In the Daisy Wiki, the publisher request set to be used can be specified in the siteconf.xml

4.12.22.1.3 p:prepareDocument

The p:prepareDocument can have an optional attribute called `inlineParts`. This attribute specifies a comma-separated list of part type names (or IDs) for which the content should be inlined. By default this only happens for parts for which the Daisy-HTML flag is set to true.

The inlining will only happen if the actual part has a mime-type that starts with "text/" or if the mime-type is recognized as an XML mime-type. Recognized XML mime-types are currently text/xml, application/xml or any mime type ending with "+xml".

If it is an XML mime-type, then the content will be parsed and inserted as XML. Otherwise, it will be inserted as character data (assuming UTF-8 encoding of the part text data). If the inlining actually happened, an attribute inlined="true" is added to the d:part element in question.

4.12.23 p:publisherRequest

p:publisherRequest is the root element of a publisher request document.

A basic, empty publisher request is structured as follows:

```
<p:publisherRequest
    xmlns:p="http://outerx.org/daisy/1.0#publisher"
    locale="en-US"
    versionMode="live"
    exceptions="throw">

 [... various publisher requests ...]

</p:publisherRequest>
```

The reponse of a publisher request is structured as follows:

```
<p:publisherResponse
    xmlns:p="http://outerx.org/daisy/1.0#publisher">

 [... responses to the various requests ...]

</p:publisherResponse>
```

About the attributes on the p:publisherRequest element:

The **locale** attribute is optional, by default the en-US locale will be used. If the publisher request is executed as part of another publisher request (see p:preparedDocuments), the locale will default to the locale of the 'parent' publisher request.

The **versionMode** attribute is optional, valid values are live (the default) or last. This attribute indicates which version of a document should be used by default if no explicit version is indicated. If its value is 'last', it will also cause the option 'search_last_version' to be set for various queries (those embedded in documents when requesting a preparedDocument, those executed by p:performQuery or p:forEach). It will also influence the version mode of the navigation tree when using p:navigationTree.

The **exceptions** attribute is also optional, *throw* is its default value. Basically this means that if an exception occurs during the processing of the publisher request, it will be thrown. It is also possible to specify *inline*, in which case the error description will be embedded in the p:publisherResponse element, but no exception will be thrown.

The **styleHint** attribute (optional, not shown above) will simply be replicated on the p:publisherResponse element. The publisher itself does not interpret the value of this attribute, it can be used by the caller of the publisher to influence the styling process. In the case of the

Daisy Wiki, the styleHint attribute can contain the name of an alternative document-styling XSL to use (instead of the default doctype-name.xsl).

4.12.24 p:resolveDocumentIds

This element allows to retrieve the names of a set of documents of which you have only the ID. The advantage compared to using simply the repository API is that this only requires one remote call for as many documents as you need (assuming you are using the remote API, otherwise it does not make a difference).

4.12.24.1 Request

Syntax:

```
<p:resolveDocumentIds branch="..." language="...">
  <p:document id="..." branch="..." language="..." version="..."/>
</p:resolveDocumentIds>
```

The branch and language attributes on the p:resolveDocumentIds element specify the default branch and language to use if it is not specified on the individual documents. These attributes are optional, the main branch and default language is used as default when these attributes are not specified.

The branch, language and version attributes on the p:document element are optional. By default the live version is used, or the last version if the document does not have a live version.

4.12.24.2 Response

The result has the following format:

```
<p:resolvedDocumentIds>
  <p:document id="..." branch="..." language="..." version="..." name="..."/>
</p:resolvedDocumentIds>
```

The id, branch, language and version attributes are simply copied from the requesting document element. The name attribute is added. The p:document elements in the result corresponds to those in the request at the same position.

If there is some error (such as the document does not exist, or the specified branch or language does not exist), an error message is put in the name attribute.

4.12.25 p:resolveVariables

Resolves variables in the specified text strings.

4.12.25.1 Request

```
<p:resolveVariables>
  <p:text>...</p:text>
  ... more p:text elements ...
</p:resolveVariables>
```

Variables should be embedded in the text using ${varname} syntax ($$ is used to escape $).

4.12.25.2 Response

```
<p:resolvedVariables>
  <p:text>...</p:text>
  ... more p:text elements ...
</p:resolvedVariables>
```

Each p:text element in the response corresponds to the p:text element in the request at the same position.

4.12.26 p:selectionList

This instruction allows to retrieve the selection list of a field type.

4.12.26.1 Request

Syntax:

```
<p:selectionList fieldType="..." branch="expr" language="expr"/>
```

The fieldType attribute can contain either the name or ID of the field type.

The branch and language attributes are optional, if not present they default to those of the context document, if any, and otherwise to the main branch and default language. The branch and language only make a difference when the selection list implementation needs them, e.g. for query selection lists with "filter variants automatically" behaviour.

4.12.26.2 Response

If the field type has no selection list, the output will contain nothing, otherwise a d:expSelectionList element will be present. The 'exp' prefix stands for 'expanded', this in contrast to the defintion of the selection list (which can e.g. be a query).

4.12.27 p:shallowAnnotatedVersion

Returns the shallow version XML, this the version XML without field and part information in it.

4.12.27.1 Request

Syntax:

```
<p:shallowAnnotatedVersion/>
```

4.12.27.2 Response.

A d:version element.

If you requested the live version of the document, but the document does not have a live version, there will simply be no d:version element in the response.

4.12.28 p:subscriptionInfo

Returns whether the user is subscribed for email notifications on the current context document.

4.12.28.1 Request

Syntax:

```
<p:subscriptionInfo/>
```

4.12.28.2 Response

The response is the same element with the actual subscription status added:

```
<p:subscriptionInfo subscribed="true|false"/>
```

4.12.29 p:variablesConfig

This is not a publisher instruction, but rather configuration information for the variable resolution.

With "variables" we mean the variables that can be embedded in Daisy-HTML parts and document names. See [todo] for more information on this topic.

The p:variablesConfig element can only occur as first child of p:publisherRequest (page 106).

4.12.29.1 Syntax

The syntax is as follows:

```
<p:variablesConfig>

  <p:variableSources>
    <p:variableDocument id="..." branch="..." language="..."/>
    [... more p:variableDocument elements ...]
  </p:variableSources>

  <p:variablesInAttributes [allAttributes="true|false"] >
     <p:element name="img" attributes="daisy-caption,alt"/>
     [... more p:element elements ...]
  </p:variablesInAttributes>

</p:variablesConfig>
```

All elements are optional.

The p:variableDocument elements point to documents containing the variable-to-value mappings. These are documents containing a part VariablesData. The XML format that should be in there is documented in the variables documents, it is not repeated here to avoid duplication.

The p:variablesInAttributes element configures whether variables should be resolved in attributes. If this element is not present, no variable resolving in attributes will happen. For the case where speed is a concern, you can configure which attributes on which elements should be considered.

4.12.29.2 Effect of p:variablesConfig

When a p:variablesConfig is specified, variable resolution will be enabled for a number of cases:

- most prominently, for p:preparedDocuments, variable resolution in Daisy-HTML parts.
- p:navigationTree: will perform variable resolution in all navigation tree node labels.
- p:performQuery: will perform variable resolution in result set columns corresponding to the 'name' identifier (at the time of this writing, this will only happen for exact selects of 'name', and not when name is used as part of more complex expressions)
- p:annotatedDocument and p:annotatedVersion: variable resolution in the document names

4.12.30 p:variablesList

Returns a list of all defined variables, according to the active p:variableConfig (page 109) of the current publisher request. This is mostly useful to let editors pick variables from the list of available variables.

4.12.30.1 Request

This request requires no attributes, so its syntax is simply:

```
<p:variablesList/>
```

4.12.30.2 Response

The response looks like this:

```
<p:variablesList>
  <p:variable name="...">...the value...</p:variable>
  ... more p:variable elements ...
</p:variablesList>
```

If there are no variables, an empty p:variablesList element will be present in the response.

4.13 Backup locking

The practical side of making backups is explained in the section Making backups (page 318). Here we only describe the backup-lock mechanism, which is of use if you want to write your own backup tool.

The repository server uses multiple storages: a relational database, the blobstore (a filesystem directory), and the full text indexes (also stored on the filesystem). Next to that, the JMS system also has its own database. Daisy does not employ some fancy distributed transaction manager with associated log, therefore it has its own simple mechanism to allow to take a consistent backup of these different stores. Daisy allows to take a "backup lock" on the repository server. This will:

- disable write operations to the blobstore (read operations still possible)
- disable receiving and sending of JMS messages
- disable updates to the full text index

- suspend any running document tasks (as these would likely start to fail if the blobstore does not allow updates)

A backup lock is requested via the JMX interface. It needs a "timeout" parameter that specifies how long to wait for any running operations to end.

While the backup lock is active, all read operations will continue to work, and update operations that only involve the repository's relational database will continue to work. Operations that require an update to the blobstore (such as saving a document in most cases) will give an exception.

4.14 Image thumbnails and metadata extraction

The repository server contains an (optional) component that can perform image thumbnailing, extraction of metadata (width, height and, for JPEG, arbitrary Exif fields), and automatic rotation of JPEG images as indicated in the Exif data. This component is registered as a "pre-save-hook", this is a component which gets called before a document is saved, and which can modify the content of the to-be-saved document.

For further reference, we will call this component the *image hook*.

The image hook can be configured to react on multiple document types, and for each document type it is possible to specify what needs to be done:

- generate a thumbnail: size and name of the part to store the thumbnail in can be configured.
- generate a preview image (somewhat larger then a thumbnail): ditto config as for thumbnails
- assign the width and the height of the image to fields of your choice (these need to be of type long)
- a mapping of Exif metadata fields to fields in the Daisy document. The mapping can bind different data types, and can bind either the raw value or a formatted value. For example, the field "Exposure Program" can be bound either as the numeric value '2' or as the string "Program normal".

By default, the image hook is configured to handle the Image document type as defined by the Daisy Wiki.

The configuration of the image hook can be adjusted via the myconfig.xml configuration file of the repository server.

The image hook will only perform its work if the part containing the original image data changed, or if any of the to-be extracted information is missing in the target document. Therefore, if for some reason you want to trigger the image hook to redo its work, you need to make sure one of these conditions is true.

4.15 Programming interfaces

The native API of the Daisy repository server is its Java interface. To allow other processes (on the same or another computer) to talk to the repository server, a HTTP+XML based interface is available. Lastly, the Java API of the Daisy repository server is also implemented in a "remote" variant, whereby it transparently uses the HTTP+XML interface to talk to the repository server.

Since a variety of scripting languages can be run on top of the Java VM, it is possible to use the Java API from such scripting languages, which is convenient for smaller jobs.

4.15.1 Java API

4.15.1.1 Introduction

Daisy is written in Java and thus its native interface is a Java API. This Java API is packaged separately, and consists of two jars:

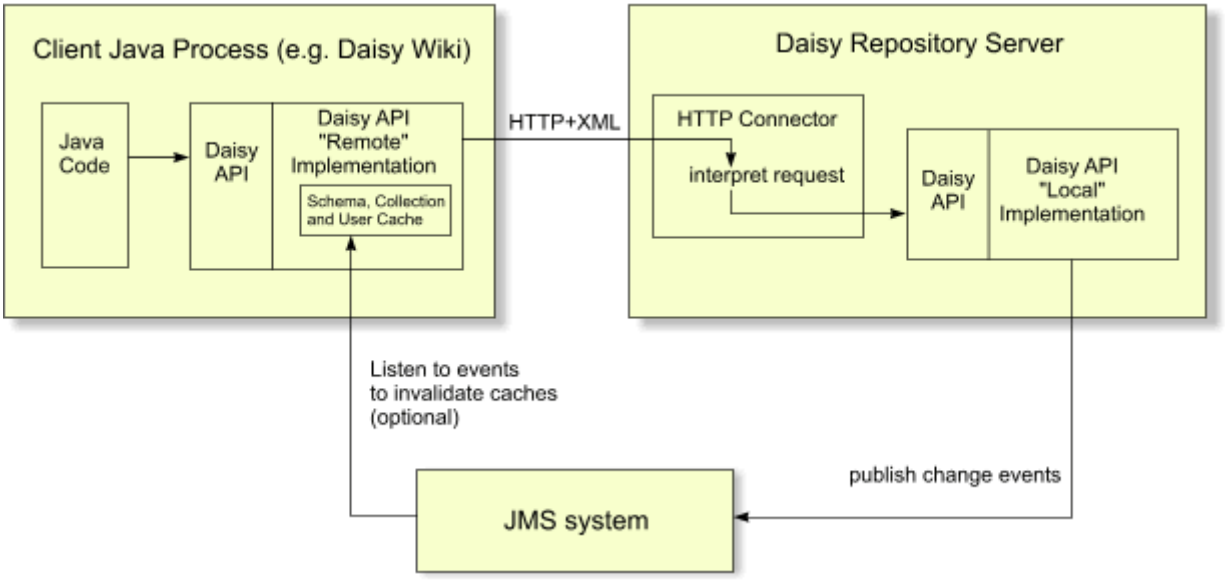
```
daisy-repository-api-<version>.jar
daisy-repository-xmlschema-bindings-<version>.jar
```

The second jar, the xmlschema-bindings, are Java classes generated from XML Schemas, and form a part of the API. To write client code that talks to Daisy, at compile you need only the above two jars in the classpath (at runtime, you need a concrete implementation, see further on).

There are two implementations of this API available:

- a *local* implementation, this is the actual implementation in the repository server, which does the real work
- a *remote* implementation, this is an implementation that talks via the HTTP+XML protocol to the repository server

This is illustrated in the diagram below.



To be workable, the remote implementation caches certain information: the repository schema (document, field and part types), the collections, and the users (needed to be able to quickly map user IDs to user names). To be aware of changes done by other clients, the remote implementation can listen to the JMS events broadcasted by the server to update these caches. This is optional, for example a short-running client application that performs a specific task probably doesn't care much about this, especially since the cached information is not the kind of information that changes frequently. Even when JMS-based cache invalidation is disabled, the caches of a certain remote implementation instance are of course kept up-to-date for changes done through that specific instance.

Examples of applications making use of the remote API implementation are the Daisy Wiki, and the installation utilities daisy-wiki-init and daisy-wiki-add-site. Especially the source of those last two can serve as useful but simple examples of how to write client applications. The shell scripts to launch them show the required classpath libraries.

4.15.1.2 Reference documentation

See the javadoc documentation[6].

4.15.1.3 Quick introduction to the Java API

The Daisy Java API is quite high-level, and thus easy-to-use. The start point to do any work is the RepositoryManager interface, which has the following method:

```
Repository getRepository(Credentials credentials) throws RepositoryException;
```

The Credentials parameter is simply an object containing the user name and password. By calling the getRepository method, you get an instance of Repository, through which you can access all the repository functionality. The obtained Repository instance is specific for the user specified when calling getRepository. The Repository object does not need to be released after use. It is a quite lightweight object, mainly containing the authentication information.

Let's have a look at some of the methods of the Repository interface.

```
Document createDocument(String name, String documentTypeName);
```

Creates a new document with the given name, and using the named document type. The document is not immediately created in the repository, to do this you need to call the save() mehod on the Document. But first you need to set all required fields and parts, otherwise the save will fail (it is possible to circumvent this, see the full javadocs).

```
Document getDocument(long documentId, boolean updateable) throws RepositoryException;
```

Retrieves an existing document, specified by its ID. If the flag 'updateable' is false, the repository will return a read-only Document object, which allows it to return a shared cached copy. In the remote implementation, this doesn't matter since it doesn't perform any caching, but in the local implementation this can make a very huge difference.

```
RepositorySchema getRepositorySchema();
```

Returns an instance of RepositorySchema, through which you can inspect and modify the repository schema (these are the document, part and field types).

```
AccessManager getAccessManager();
```

Returns an instance of AccessManager, through which you can inspect and modify the ACL, and get the ACL evaluation result for a certain document-user-role combination.

```
QueryManager getQueryManager();
```

Returns an instance of QueryManager, through which you can perform queries on the repository using the Daisy Query Language.

```
CollectionManager getCollectionManager();
```

Returns an instance of CollectionManager, through which you can create, modify and delete document collections.

```
UserManager getUserManager();
```

Returns an instance of UserManager, through which you can create, modify and delete users.

The above was just to give a broad idea of the functionality available through the API. For more details, consult the complete JavaDoc of the API.

4.15.1.4 Writing a Java client application

Let's now look at a practical example.

Here's a list of jars you need in the CLASSPATH to use the remote repository API implementation.

This list is bound to change in different Daisy releases. We advise you to copy the settings of the CLASSPATH defined in the daisy-js (daisy-js.bat on Windows) script. (possibly removing the 'rhino' and 'daisy-javascript' jars)
The list below was last updated for Daisy 1.5.1:

```
DAISY_HOME/lib/daisy/jars/daisy-repository-api-1.5.jar
DAISY_HOME/lib/daisy/jars/daisy-repository-client-impl-1.5.jar
DAISY_HOME/lib/daisy/jars/daisy-repository-spi-1.5.jar
DAISY_HOME/lib/daisy/jars/daisy-util-1.5.jar
DAISY_HOME/lib/avalon-framework/jars/avalon-framework-api-4.1.5.jar
DAISY_HOME/lib/daisy/jars/daisy-repository-common-impl-1.5.jar
DAISY_HOME/lib/commons-httpclient/jars/commons-httpclient-2.0.2.jar
DAISY_HOME/lib/xmlbeans/jars/xbean-2.1.0.jar
DAISY_HOME/lib/xmlbeans/jars/xmlpublic-2.1.0.jar
DAISY_HOME/lib/stax/jars/stax-api-1.0.jar
DAISY_HOME/lib/daisy/jars/daisy-repository-xmlschema-bindings-1.5.jar
DAISY_HOME/lib/concurrent/jars/concurrent-1.3.2.jar
DAISY_HOME/lib/commons-logging/jars/commons-logging-1.0.4.jar
DAISY_HOME/lib/commons-collections/jars/commons-collections-3.1.jar
DAISY_HOME/lib/daisy/jars/daisy-jmsclient-api-1.5.jar
DAISY_HOME/lib/geronimo-spec/jars/geronimo-spec-jms-1.1-rc4.jar

(below only if you need the htmlcleaner)

DAISY_HOME/lib/daisy/jars/daisy-htmlcleaner-1.5.jar
DAISY_HOME/lib/nekohtml/jars/nekohtml-0.9.5.jar
DAISY_HOME/lib/nekodtd/jars/nekodtd-0.1.11.jar
DAISY_HOME/lib/xerces/jars/xercesImpl-2.6.2.jar
DAISY_HOME/lib/xerces/jars/xmlParserAPIs-2.2.1.jar
```

So depending on your own habits, you could set up a project in your IDE with these jars in the classpath, or make an Ant project, or whatever.

Below a simple and harmless example is shown: performing a query on the repository.

```
package mypackage;

import org.outerj.daisy.repository.RepositoryManager;
import org.outerj.daisy.repository.Credentials;
import org.outerj.daisy.repository.Repository;
import org.outerj.daisy.repository.query.QueryManager;
import org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager;
import org.outerx.daisy.x10.SearchResultDocument;

import java.util.Locale;
```

```
public class Search {
    public static void main(String[] args) throws Exception {
        RepositoryManager repositoryManager = new RemoteRepositoryManager(
            "http://localhost:9263", new Credentials("guest", "guest"));
        Repository repository =
            repositoryManager.getRepository(new Credentials("testuser", "testuser"));
        QueryManager queryManager = repository.getQueryManager();

        SearchResultDocument searchresults =
            queryManager.performQuery("select id, name where true",
Locale.getDefault());
        SearchResultDocument.SearchResult.Rows.Row[] rows =
            searchresults.getSearchResult().getRows().getRowArray();

        for (int i = 0; i < rows.length; i++) {
            String id = rows[i].getValueArray(0);
            String name = rows[i].getValueArray(1);
            System.out.println(id + " : " + name);
        }

        System.out.println("Total number: " + rows.length);

    }
}
```

The credentials supplied in the constructor of the RemoteRepositoryManager specify a user to be used for filling the caches in the repository client. This can be any user, the user doesn't need any special access privileges.

4.15.1.5 Java client application with Cache Invalidation

FIXME

Needs updating for switch to ActiveMQ (java sample code also need updating due to new required JMS client ID)

For long-running client applications you may want to have the caches of the client invalidated when changes happen by other users. For a code sample of how to create a JMS client and pass it on to the RemoteRepositoryManager, see JMS Cache Invalidation Sample (page 0).

For this example to run, you'll need the JMS client implementation jars in the CLASSPATH, in addition to the earlier listed jars:

```
DAISY_HOME/lib/daisy/jars/daisy-jmsclient-impl-1.3.jar
DAISY_HOME/lib/exolabcore/jars/exolabcore-0.3.7.jar
DAISY_HOME/lib/openjms/jars/openjms-client-0.7.6.jar
```

4.15.1.6 More

It might be interesting to also have a look at the notes on scripting using Javascript (page 116), since there essentially the same API is used from a different language.

4.15.2 Scripting the repository using Javascript

4.15.2.1 Introduction

Rhino, a Java-based Javascript implementation, makes it easy to use the Java API of the repository server to automate all kinds of operations. In other words: easy scripting of the repository server. It brings all the benefits of Daisy's high-level repository API without requiring Java knowledge or the setup of a development environment.

4.15.2.2 How does it work?

1. Write a Javascript, save it in a ".js" file.
2. Open a command prompt or shell, set the DAISY_HOME environment variable to point to your Daisy installation
3. Go to the directory <*DAISY_HOME*>/bin
4. Execute "daisy-js <*name-of-scriptfile*>"

4.15.2.3 Connecting to the repository server from Javascript

The basic code you need to connect to the repository server from Javascript is the following:

```
importPackage(Packages.org.outerj.daisy.repository);
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                    new Credentials("guest", "guest"));
var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));
```

Some explanation:

The *importPackage* and *importClass* statements are used to make the Daisy Java API available in the Javascript environment.

Then a *RepositoryManager* is constructed, this is an object from which Repository objects can be retrieved. A *Repository* object represents a connection to the Daisy Repository Server for a certain user. Typically, you only construct one RepositoryManager, and then retrieve different Repository objects from it if you want to perform actions under different users.

The first argument of the RepositoryManager constructor is the address of the HTTP interface of the repository server (9263 is the default port). The second argument is a username and password for a user that is used inside the implementation to fill up caches. This can be any user, here we've re-used the guest user of the Wiki. This user does not need to have any access permissions on documents. (Inside the implementation, some often needed info like the repository schema and the collections is cached)

Then from the RepositoryManager a Repository for a specific user is retrieved.

4.15.2.4 Repository Java API documentation

Reference documentation of the Daisy API is available online[7] and included in the binary distribution in the *apidocs* directory (open the file index.html in a web browser). See also Java API (page 112).

4.15.2.5 Examples

4.15.2.5.1 Creating a document (uploading an image)

This example uploads an image called "myimage.gif" from the current directory into the repository.

```
importPackage(Packages.org.outerj.daisy.repository);
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                    new Credentials("guest", "guest"));
var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));

var document = repository.createDocument("My test image", "Image");
var imageFile = new java.io.File("myimage.gif");
document.setPart("ImageData", "image/gif", new FilePartDataSource(imageFile));
document.save();

print("Document created, ID = " + document.getId());
```

See the API documentation for the purpose of the arguments of the methods. For example, the text "Image" supplied as the second argument of the createDocument method is the name of the document type to use for the document. Likewise, the first argument of setPart, "ImageData", is the name of the part.

It would be an interesting exercise to extend this example to upload a whole directory of images :-)

4.15.2.5.2 Performing a query

```
importPackage(Packages.org.outerj.daisy.repository);
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);
importPackage(Packages.java.util);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                    new Credentials("guest", "guest"));
var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));
var queryManager = repository.getQueryManager();

var searchresults = queryManager.performQuery("select id, name where true",
Locale.getDefault());
var rows = searchresults.getSearchResult().getRows().getRowArray();
for (var i = 0; i < rows.length; i++) {
  print(rows[i].getValueArray(0) + " : " + rows[i].getValueArray(1));
}

print("Total number: " + rows.length);
```

4.15.2.5.3 Creating a user

```
importPackage(Packages.org.outerj.daisy.repository);
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                    new Credentials("guest", "guest"));
var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));

// With the UserManager we can manage users and roles
var userManager = repository.getUserManager();
```

```
// Get references to some roles, we'll need them in a moment
var guestRole = userManager.getRole("guest", false);
var adminRole = userManager.getRole("Administrator", false);

// check if current user has admin role, and exit if not
var me  = userManager.getUser(repository.getUserId(), false);
var adminRoleId = adminRole.getId();
if (!me.hasRole(adminRoleId))
{
  print ("current user lacks admin rights to add new user. ");
  exit;
}
repository.switchRole(adminRoleId);

// Create the new user
var newUser = userManager.createUser("john");

// The user needs to be added to at least one role
newUser.addToRole(guestRole);
newUser.addToRole(adminRole);

// Optionally, set a default role which will be active after
// logging in. If not set, all roles (with the exception of
// the Administrator role) will be active on login
// newUser.setDefaultRole(guestRole);

// Password is required when using Daisy's built-in authentication scheme
newUser.setPassword("boe");

// Alternatively, set another authentication scheme:
// newUser.setAuthenticationScheme("my-ldap");

// Optional things
newUser.setFirstName("John");
newUser.setLastName("Johnson");

// Setting updateableByUser will allow the user to access the
// user settings page in the Wiki, so that the user can
// update here e-mail
newUser.setUpdateableByUser(true);

newUser.save();
```

4.15.2.5.4 Changing the password of an existing user

```
importPackage(Packages.org.outerj.daisy.repository);
importPackage(Packages.org.outerj.daisy.repository.user);
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                    new Credentials("guest", "guest"));
var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));
repository.switchRole(Role.ADMINISTRATOR);

var userManager = repository.getUserManager();
var user = userManager.getUser("someuser", true);
user.setPassword("somepwd");
user.save();
```

4.15.2.5.5 Workflow samples

See Workflow Java API (page 295).

4.15.2.5.6 Your example here

If you've got a cool example to contribute, just write to the Daisy mailing list.

4.15.3 HTTP API

4.15.3.1 Introduction

Daisy contains a HTTP+XML interface, which is an interface to talk to the repository server by exchanging XML messages over the HTTP protocol. This interface offers full access to all functionality of the repository.

The HTTP protocol is a protocol that allows to perform a limited number of methods (Daisy uses GET, POST and DELETE) on an unlimited number of resources, which are identified by URIs. The GET method is used to retrieve a representation of the addressed resource, POST to trigger a process that modifies the addressed resource, and DELETE to delete a resource.

With HTTP, all calls are independent of each other, there is no session with the server.

The Daisy HTTP interface listens by default on port 9263. You can easily try it out, for example if Daisy is running on your localhost, just enter the URL below in the location bar of the browser, and press enter. The browser will then send a GET request to the server. The example given here is a request to execute a query (written in the Daisy Query Language). This request doesn't require an XML payload, all parameters are specified as part of the URL. Note that spaces in an URL must be encoded with a plus symbol.

```
http://localhost:9263/repository/query?q=select+id,name+where+true&locale=en
```

The browser will ask a user name and password, enter your Daisy repository username and password (e.g., the one you otherwise use to log in on the Daisy Wiki), or use the user name "guest" and password "guest" (only works if you installed the Daisy Wiki). The browser will show the XML response received from the server (in some browsers, you might need to do "view source" to see it).

Not all operations can be performed as easily as the above example: some require POST or DELETE as method, some require an XML document in the body of the request, and some even require a multipart-formatted request body (the document create and update operations, which need to upload the binary part data next to the XML message). If you have a programming language with a decent HTTP client library, none of this should be a problem.

4.15.3.2 Authentication

All requests require authentication. Authentication is done using BASIC authentication.

If you want to log in as another role then the default role of a user, append "@<roleid>" to the login (without the quotes). Note that it must be the id of the role, not its name. For example, if your default role is not Administrator (ID: 1), but you would like to perform the request as Administrator, and your login is "jules", you would use "jules@1". When the login itself contains an @-symbol, it must be escaped by doubling it (i.e. each @ should be replaced with @@). Multiple active roles can be specified using a comma-separated list, e.g. "jules@1,105".

4.15.3.3 Robustness

The current implementation doesn't do (many) checks on the XMLs posted as part of a HTTP request. This means that for example missing elements or attributes might simply cause little-descriptive (but harmless) "NullPointerExceptions" to occur.

The reason for this is that we use the HTTP API mostly via the repository Java client, which generates valid messages for us.

Since the XML posted to a resource is usually the same as the XML retrieved via GET on the same resource, it is easy to get examples of correct XML messages. XML Schemas are also available (see further on), though being schema-valid doesn't necessarily imply the message is correct.

4.15.3.4 Error handling

If a response was handled correctly, the server will answer with HTTP status code 200 (OK). If the status code has another value, it means something went wrong.

For errors generated explicitly, or when a Java exception occurs, an XML message is created describing the exception, and is returned with a status code 202 (Accepted). The XML message consists of an <error> root element, with as child either a <description> element or a <cause> element. The <description> element contains a simple string describing the error. The <cause> element is used in case a Java exception was handled, and contains further elements describing the exception (including stacktrace), and can include <cause> elements recursively describing the "causing" exceptions of that exception. To see an example of this, simply do a request for a non-existing resource, e.g.:

```
http://localhost:9263/repository/document/99999999
```

(assuming there is no document with ID 99999999)

When executing a method (GET, POST, DELETE, ...) on a resource that doesn't support that method you will get status code 405 (Method Not Allowed).

Incorrect or missing authentication information will give status code 401 (Unauthorized).

Missing request parameters, or invalid ones (e.g. giving a string where a number was expected) will give status code 400 (Bad Request).

Doing a request for a non-existing resource will give status code 404 (Not Found)

4.15.3.5 Intro to the reference

The rest of this document describes the available URLs, the operations that can be performed upon them, and the format of the XML messages. The descriptions can be dense, the current goal of this document is just to give a broad overview, more details might be added later. You can always ask for more information on the Daisy Mailing List.

You can also investigate how things are supposed to work by monitoring the HTTP traffic between the Daisy Wiki and the Daisy Repository Server.

Sometimes XML Schema files are referenced, these can be found in the Daisy source distribution.

4.15.3.6 Core Repository Interface

4.15.3.6.1 Documents

On many document-related resources, request parameters called branch and language can be added (this will be mentioned in each case). The value of these parameters can be either a name or ID of a branch or language. If not specified, the branch "main" and the language "default" are assumed.

4.15.3.6.1.1 /repository/document

This resource represents the set of all documents. GET is not supported on this resource (you can retrieve a list of all documents using a query).

POST on this resource is used to create a new document, which also implies the creation of a document variant, since a document cannot exist without a document variant. The payload should be a multipart request having one multipartrequest-part (we use this long name to distinguish with Daisy's document parts) containing the XML description of the new document, and other multipartrequest-parts containing the content of the document parts (if any). The multipartrequest-part containing the XML should be called "xml", and should conform to the document.xsd schema. The part elements in the XML should have dataRef attributes whose value is the name of the multipartrequest-part containing the data for that part.

The server will return the XML description of the newly created document as result. This XML will, among other things, have the id attribute completed with the ID of the new document. Example scenario: creating a new document

This example illustrates how to create a new document in the repository over the HTTP interface using the curl[8] tool. Curl is a handy command-line tool to do HTTP (and other) requests, and is standard available on many Linux distributions (it exists for Windows too).

Suppose we want to create a new document of type 'SimpleDocument' (as used in the Daisy Wiki), with the part 'SimpleDocumentContent'. We start by creating the XML description of the document, and save it in a file called newdoc.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:document
  xmlns:ns="http://outerx.org/daisy/1.0"
  name="My test doc"
  typeId="2"
  owner="3"
  validateOnSave="true"
  newVersionState="publish"
  retired="false"
  private="false"
  branchId="1"
  languageId="1"
  >
  <ns:customFields/>
  <ns:collectionIds>
    <ns:collectionId>1</ns:collectionId>
  </ns:collectionIds>
  <ns:fields/>
  <ns:parts>
    <ns:part mimeType="text/xml" typeId="2" dataRef="data1"/>
  </ns:parts>
  <ns:links/>
</ns:document>
```

Some items in the above XML will need to be changed for your installation:

- document/@typeId: this is the ID of the document type to use for the document. In my installation the ID for 'SimpleDocument' is 2, but you need to check that on your installation. You can see this either on the administration pages of the Daisy Wiki, or by doing a request to http://localhost:9263/repository/schema/documentType
- document/@owner: this is the ID of the owner of the document, this should be an existing user ID. You can again see the user IDs on the administration pages of the Daisy Wiki.
- document/parts/part/@typeId: this is the ID of the part type. In my installation the ID for 'SimpleDocumentContent' is 2, but you need to check that on your installation. You can see this either on the administration pages of the Daisy Wiki, or by doing a request to http://localhost:9263/repository/schema/partType

Now we need to create a file containing the content of the part we're going to add. For example create a file called 'mynewfile.xml' and put the following in it:

```
<html>
  <body>

    <h1>Hi there!</h1>

    <p>This is a test document.</p>

  </body>
</html>
```

Finally we are ready to create the document using curl:

```
curl --basic
     --user testuser:testuser
     --form xml=@newdoc.xml
     --form data1=@mynewfile.xml
     http://localhost:9263/repository/document
```

You need to enter all arguments on one line of course, and change user, password and server URLs as appropriate for your installation. Note that the form parameter 'data1' corresponds to the dataRef attribute in the newdoc.xml file (you can choose any name you want for these, if you have multiple parts use different names)

4.15.3.6.1.2 /repository/document/<id>

<id> should be replaced with the ID of an existing document.
Retrieving a document

GET on this resource retrieves an XML description of a document, with a certain variant of the document. The XML will contain the data of the most recent version of the document variant. The (binary) part data is not embedded in the XML, but must be retrieved separately using the following URL (described further on):

```
/repository/document/<document-id>/version/<version-id>/part/<parttype-id>/data
```

To specify the document variant, add the optional request parameters branch and language.
Creating a document or adding a document variant

POST on this resource is used to update a document (and/or document variant), or to add a new variant to it. When adding a new variant there are two possibilities: initialise the new variant with the content of an existing variant, or create a new variant from scratch. We now describe these three distinct cases.

To update an existing document (document variant), the format of the POST is similar as when creating a document, that is, it should contain a multipart-format body. The XML in this case should be an updated copy of the XML retrieved via the GET on this resource. Unmodified parts don't need to be uploaded again.

To create a new variant from scratch, again the POST data is similar as when creating a new document. In addition, three request parameters must be specified:

- createVariant with the value yes
- startBranch
- startLanguage

Although the variant is created from scratch, it is only possible to add a new variant to a document if you have at least read access to an existing variant. The new variant to be created is specified by the branchId and languageId attributes within the posted XML.

Creating a new variant based on an existing variant is rather different. In this case no XML body or multipart-request must be done, but a POST operation with the following request parameters:

- startBranch
- startLanguage
- startVersion: specify -1 for last version, -2 for live version
- newBranch
- newLanguage

These parameter names explain themselves I think. The branches and languages can be specified either by name or ID.
Deleting a document or a document variant

DELETE on this resource permanently deletes the document. This will delete the document and all its variants.

To delete only one variant of the document, specify the request parameters branch and language.

4.15.3.6.1.3 /repository/document/<id>/version

GET on this resource returns a list of all versions in a document variant as XML. For each version, only some basic information is included (the things typically needed to show a version overview page).

To specify the document variant, add the optional request parameters branch and language.

4.15.3.6.1.4 /repository/document/<id>/version/<id>

GET on this resource returns the full XML description of this version. As when requesting a document, the actual binary part data is not embedded in the XML but has to be retrieved separately.

POST on this resource is used to modify the version state (which is the only thing of a version that can be modified, other then that, versions are read-only once created). The request should have two parameters:

- action=changeState

- newState=publish|draft

For both the GET and POST methods, add the optional request parameters branch and language to specify the variant.

4.15.3.6.1.5 /repository/document/<id>/version/<id>/part/<id>/data

GET on this resource retrieves the data of a part in a certain version of a document. The meaning of the <id>'s is as follows:

1. The first <id> is the document ID
2. The second <id> the version ID (1, 2, 3, ...) or the strings "live" or "last" to signify the live or last version
3. The third <id> is the part type ID or the part type name of the part to be retrieved (if the first character is a digit, it is supposed to be an ID. Part type names cannot begin with a digit).

To specify the document variant, add the optional request parameters branch and language.

On the response, the HTTP headers Last-Modified, Content-Length and Content-Type will be specified.

4.15.3.6.1.6 /repository/document/<id>/lock

See the lock.xsd file for the XML Schema of the XML used to interact with this resource.

GET on this resource returns information about the lock, if any.

POST on this resource is used to create a lock. In this case, all attributes in the XML must have a value except for the hasLock attribute.

DELETE on this resource is used to remove the lock (if any). No request body is required.

For all three methods, the returned result is the XML description of the lock after the performed operation (possibly describing that there is no lock).

A lock applies to a certain variant of a document. To specify the document variant, add the optional request parameters branch and language.

4.15.3.6.1.7 /repository/document/<id>/comment

See comment.xsd for the XML Schema of the messages.

GET on this resource returns the list of comments for a document variant. To specify the document variant, add the optional request parameters branch and language.

POST on this resource creates a new comment. The branch and language are in this case specified in the XML message.

4.15.3.6.1.8 /repository/document/<id>/comment/<id>

DELETE on this resource deletes a comment. The second <id> is the ID of the comment. To specify the document variant, add the optional request parameters branch and language.

Other methods are not supported on this resource.

4.15.3.6.1.9 /repository/document/<id>/availableVariants

A GET on this resource returns the list of variants that exist for this document.

4.15.3.6.2 Schema Management

4.15.3.6.2.1 /repository/schema/(part|field|document)Type

These resources represent the set of part, field and document types.

POST to these resources is used to create a new part, field or document type. The request body should contain an XML message conforming to the schemas found in fieldtype.xsd, parttype.xsd or documenttype.xsd.
Example scenario

This example illustrates how to create a new field type (with a selection list), simply by using the well-known "wget" tool.

This is the XML that we'll send to the server:

```
<?xml version="1.0"?>
<fieldType name="myNewField" valueType="string" deprecated="false"
           aclAllowed="false" size="0"
           xmlns="http://outerx.org/daisy/1.0">
  <labels>
    <label locale="">My New Field</label>
  </labels>
  <descriptions>
    <description locale="">This is a test field</description>
  </descriptions>
  <selectionList>
    <staticSelectionList>
      <listItem>
        <labels/>
        <string>value 1</string>
      </listItem>
      <listItem>
        <labels/>
        <string>value 2</string>
      </listItem>
    </staticSelectionList>
  </selectionList>
</fieldType>
```

Let's say we save this in a file called newfieldtype.xml. We can then create the field by executing:

```
wget --post-file=newfieldtype.xml
     --http-user=testuser@1
     --http-passwd=testuser
     http://localhost:9263/repository/schema/fieldType
```

This supposes that "testuser" exists and has the Administrator role, which is required for creating field types.

Wget will save the response from the server in a file called "fieldType". The response is the same XML but now with some additional attributes such as the assigned ID. The response XML isn't pretty formatted, if you have libxml installed you can view it pretty using:

```
xmllint --format fieldType
```

4.15.3.6.2.2 /repository/schema/(part|field|document)Type/<id>

GET on these resources retrieves the XML representation of a part, field or document type.

POST on these resources updates a part, field or document type. The request body should then contain an altered variant of the XML retrieved via GET.

DELETE on these resources deletes them. Note that deleting types is only possible if they are not in use any more by any version of any document.
Example scenario

Let's take the previous field type example again, and add an additional value to the selection list. We first retrieve the XML for the field type (check the XML response of the previous sample to know the ID of the created field type):

```
wget http://localhost:9263/repository/schema/fieldType/1
```

This will save a file called "1" (if that was the requested ID). To make it easier to work with, do a:

```
xmllint --format 1 > updatedfieldtype.xml
```

and change the updatedfieldtype.xml with an additional value in the selection list:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:fieldType xmlns:ns="http://outerx.org/daisy/1.0" size="0" updateCount="1"
  aclAllowed="false" deprecated="false" valueType="string"
  name="myNewField" lastModifier="3"
  lastModified="2004-09-09T09:06:51.032+02:00" id="1">
  <ns:labels>
    <ns:label locale="">My New Field</ns:label>
  </ns:labels>
  <ns:descriptions>
    <ns:description locale="">This is a test field</ns:description>
  </ns:descriptions>
  <ns:selectionList>
    <ns:staticSelectionList>
      <ns:listItem>
        <ns:labels/>
        <ns:string>value 1</ns:string>
      </ns:listItem>
      <ns:listItem>
        <ns:labels/>
        <ns:string>value 2</ns:string>
      </ns:listItem>
      <ns:listItem>
        <ns:labels/>
        <ns:string>value 3</ns:string>
      </ns:listItem>
    </ns:staticSelectionList>
  </ns:selectionList>
</ns:fieldType>
```

And then do:

```
wget --post-file=updatedfieldtype.xml
     --http-user=testuser@1
     --http-passwd=testuser
     http://localhost:9263/repository/schema/fieldType/1
```

4.15.3.6.2.3 /repository/schema/(part|field|document)TypeByName/<name>

GET on this resource retrieves a part, field or document type by its name.

You cannot POST on this resource, to update the type, use the previous (ID-based) resource.

4.15.3.6.2.4 /repository/schema/fieldType/<id>/selectionListData

GET on this resource retrieves the data of the selection list of the field type. If the field type does not have a selection list, an error will be returned.

When retrieving the XML representation of a field type, the selection list contained therein is the definition of the selection list, which can e.g. be a query. This resource instead returns the "expanded" selection list data, i.e. with queries executed etc.

Request parameters:

- locale: required (e.g. en-US)
- branch: defaults to main if absent
- language: defaults to default if absent

The branch and language parameters are not always important, they are used in case of selection lists that filter their items according to the branch and language.

4.15.3.6.3 Access Control Management

4.15.3.6.3.1 /repository/acl/<id>

...

4.15.3.6.3.2 /repository/filterDocumentTypes

...

4.15.3.6.4 Collection Management

4.15.3.6.4.1 /repository/collection/<id>

...

4.15.3.6.4.2 /repository/collectionByName/<name>

...

4.15.3.6.4.3 /repository/collection

...

4.15.3.6.5 User Management

4.15.3.6.5.1 /repository/user

...

4.15.3.6.5.2 /repository/role

...

4.15.3.6.5.3 /repository/user/<id>

...

4.15.3.6.5.4 /repository/role/<id>

...

4.15.3.6.5.5 /repository/userByLogin/<login>

...

4.15.3.6.5.6 /repository/roleByName/<name>

...

4.15.3.6.5.7 /repository/usersByEmail/<email>

...

4.15.3.6.5.8 /repository/userIds

...

4.15.3.6.5.9 /repository/publicUserInfo

...

4.15.3.6.5.10 /repository/publicUserInfo/<id>

...

4.15.3.6.5.11 /repository/publicUserInfoByLogin/<login>

...

4.15.3.6.6 Variant Management

4.15.3.6.6.1 /repository/branch

...

4.15.3.6.6.2 /repository/branch/<id>

...

4.15.3.6.6.3 /repository/branchByName/<name>

...

4.15.3.6.6.4 /repository/language

...

4.15.3.6.6.5 /repository/language/<id>

...

4.15.3.6.6.6 /repository/languageByName/<name>

...

4.15.3.6.7 Querying

4.15.3.6.7.1 /repository/query

GET on this resource is used to perform queries using the Daisy Query Language (page 63).

Required parameters:

- q=<some daisy query>
- locale=en-US : the locale to be used to format the result data (influences date and number formatting)

Optional parameters:

- returnKeys=true|false : false by default, if true will returns only the keys of the document variants satisfying the query (that is: the triple document ID, branch, language), instead of full results. (Remember that the result of a query is a set of document variants).
- extraCondition=<conditional expression> : will combine this conditional expression with the existing where conditions of the given query, using AND. For example, this is used in the Daisy Wiki to limit query results to a certain collection. The extraCondition is then something like: InCollection('mycollection'). Note that this is different from simply appending "AND" and the conditional expression to the end of the query, since the query can include order by and other clauses at the end.

4.15.3.6.7.2 /repository/facetedQuery

Used to perform a query for which the result contains the distinct values for the different items returned by the query. This allows to build a "faceted navigation" front end.

The query parameters are specified in an XML document which should be posted to this resource. The format of the XML is defined in facetedquery.xsd.

4.15.3.6.8 Namespace management

4.15.3.6.8.1 /repository/namespace

Use GET to get a list of all namespaces registered in this repository.

To create a namespace, use POST with parameters name and fingerprint (no XML body). Fingerprint is optional, if not specified the repository server will generate a fingerprint.

4.15.3.6.8.2 /repository/namespace/<id>

Use GET to retrieve information about a namespace, use DELETE to unregister a namespace.

The <id> is the internal ID of this namespace, which is repository-specific. It is more common to use /repository/namespaceByName

4.15.3.6.8.3 /repository/namespaceByName/<name>

Use GET to retrieve information about a namespace, use DELETE to unregister a namespace.

4.15.3.6.9 Other

4.15.3.6.9.1 /repository/userinfo

GET on this resource returns some information about the authenticated user. Takes no parameters.

4.15.3.6.9.2 /repository/comments

Usually comments are retrieved via the document they belong to, but it is also possible to get all comments of a user by doing a GET on this resource. Parameters:

- **visibility** (optional): one of: public, private, editors. Causes only comments with the given visibility to be included.

4.15.3.7 Navigation Manager Extension

See also navigation (page 166).

In Daisy 2.0, the URLs for the navigation manager changed a little bit (only the URL, not their implementation), to be consistent with the "/namespace/*" format. However, the old URLs stay supported for now. It is recommended to change them to the new ones though: /navigation -> /navigation/tree, /navigationLookup -> /navigation/lookup, /navigationPreview -> /navigation/preview

/navigation/tree

GET on this resource retrieves a navigation tree (customised for the authenticated user).

Parameters, all required unless indicated otherwise:

- navigationDocId
- navigationDocBranch: branch ID or name
- navigationDocLanguage: language ID or name
- All or none of the following:
  - activeDocumentId: ID of the document to be selected
  - activeDocumentBranch: branch ID or name
  - activeDocumentLanguage: language ID or name

- activePath: suggested path in the navigation tree where to look for the activeDocument (the same document might appear in multiple locations in the tree) (not required)
- contextualized=true|false: if true, only nodes leading to the active document will be opened, others will be closed (thus, their children will not be included in the navigation tree output)
- handleErrors=true|false: if false, when an exception occurs, the result status code from the server will not be "200 OK" but "202 Accepted" with the body containing an XML representation of the Java exception. If true and exception occurs, a normal 200 response will be given and the body will contain <n:navigationTree><n:navigationTreeError/>< /n:navigationTree> , where the "n" prefix maps to the navigation tree result namespace. The navigationTreeError element is currently empty, but might in the future contain a description of the error. The handleErrors=true parameter is useful to avoid that the rendering of a page fails completely when generating the navigation tree fails.

/navigation/preview

This resource allows to generate a navigation tree from a navigation source description specified as part of the request. This is used in the Daisy Wiki application to try out navigation trees before saving them.

Parameters:

- navigationXml: the navigation tree input XML
- branch
- language

/navigation/lookup

Resolves a path against the navigation tree and returns the result of that lookup as an XML message. For more details, see the Java API (e.g. the class NavigationLookupResult).

Required parameters:

- navigationDocId
- navigationDocBranch
- navigationDocLanguage
- navigationPath

4.15.3.8 Publisher Extension

The purpose of the Publisher Extension component is to return in one call all the data needed to publish pages in the Daisy Wiki (or other front end applications).

/publisher/request

To this resource you can POST a publisher request. A publisher request takes the form of an XML document, and is described in detail over here (page 90).

/publisher/blob

GET on this resource retrieves a blob (the data of a part).

Required parameters:

- documentId
- branch
- language
- version: a version ID, or "last" or "live"
- partType: ID or name

The last modified, content type and content length headers are set.

4.15.3.9 Email Notifier Extension

The Email Notifier extension component makes available resources for managing the email subscriptions.

/emailnotifier/subscription/<id>

<id> is the ID of a user.

GET, POST, DELETE supported. XML Schema see subscription.xsd

/emailnotifier/subscription

GET on this resource returns all the subscriptions.

/emailnotifier/(document|schema|user|acl|collection)EventsSubscribers

GET on this resource returns all subscribers for the kind of event as specified in the request path. The returned information for each subscriber includes the user ID and the locale for the subscription.

In the case of documentEventSubscribers, the following additional request parameters are required:

- documentId
- branch
- language
- collectionIds: comma separated list of IDs of collections the document belongs to.

The returned subscribers are then those that are explicitly subscribed for changes to that document, or those who are subscribed to a collection to which the document belongs, or those that are subscribed to all collections.

/emailnotifier/documentSubscription/<documentId>

This resource allows to manage document-based subscriptions without having to go through the full subscriptions.

Using POST on this resource allows to add or remove a subscription for the specified document for some user. Required parameters:

- action: add or delete
- userId
- branch (an ID)
- language (an ID)

Using DELETE on this resource removes the subscriptions for the specified document for all users. This is useful to cleanup subscriptions when a document gets deleted. If the branch and language parameters are missing, the subscriptions for all variants fo the document will be removed, otherwise only for the specified variant.

/emailnotifier/documentSubscription/<documentId>/<userId>

This resource allows to quickly check if a user is subscribed for notifications to a certain document (using GET). Request parameters branch and language are required, specifying the ID of the branch and language.

/emailnotifier/collectionSubscription/<collectionId>

Only DELETE is supported on this resource, and deletes all subscriptions for the specified collection for all users.

4.15.3.10 Emailer Extension

The emailer extension allows to send emails. Only Administrators can do this.

/emailer/mail

A POST to this resource will send an email, the following parameters are required:

- to
- subject
- messageText

4.15.3.11 Document Task Manager Extension

See also Document Task Manager (page 90).

/doctaskrunner/task

A GET on this resource retrieves all existing tasks for the current user, or the tasks of all users if the role is Administrator.

A POST on this resource is used to create a new task, in which case the body must contain an XML document describing the task (see also taskdescription.xsd).

/doctaskrunner/task/<id>

A GET on this resource retrieves information about a task.

A POST on this resource in combination with a request parameter "action" with value "interrupt" interrupts a task.

A DELETE on this resource deletes the persistent information about this task.

/doctaskrunner/task/<id>/docdetails

A GET on this resource retrieves detailed information about the execution of the task on the documents.

4.16 Extending the repository

This section contains information for people who want to plug in custom Java-based components in the repository server.

4.16.1 Repository plugins

Daisy provides a number of interfaces through which the repository functionality can be extended or customized. The components that do this are called plugins.

4.16.1.1 Anatomy of a plugin

A plugin is basically an implementation of a certain plugin interface. The various plugin interfaces are listed further on.

To deploy the plugin in the repository server, it should be packaged as a *container jar* for the Daisy Runtime (page 141). A container jar is a jar file containing a Spring container definition.

The Spring container definition should contain a bean which (usually upon initialization) registers the plugin implementation with the plugin registry (page 136). The bean which performs the registration can simply be the plugin implementation itself.

To gain access to the plugin registry, the Spring container can use the `daisy:import-service` instruction (a custom tag provided by the Daisy Runtime).

When the Spring container of the plugin is destroyed, it should properly unregister the plugin.

To deploy the plugin (page 136), the container jar can be copied to a specific subdirectory of the repository data directory.

This may sound like a lot, but it's not, as is illustrated by this example (page 137).

4.16.1.2 Plugin types

4.16.1.2.1 Extensions

Repository extensions are very generic plugins which can implement any sort of functionality. The main advantage of putting this functionality in the form of a repository extension is that the extension can then be easily retrieved using the Repository.getExtension(name) function.

Various non-core functionality in Daisy has been added as extensions, such as the NavigationManager, the Publisher, the EmailNotifier, etc.

For details see repository extensions (page 137).

4.16.1.2.2 Pre-save hooks

A pre-save hook is a component which can modify the content of a document right before it is saved. An example is the image pre-save hook included with Daisy, which can generate thumbnails in document parts and extract EXIF data to document fields.

A pre-save hook should implement the following interface:

org.outerj.daisy.repository.spi.local.PreSaveHook[9]

4.16.1.2.3 Authentication schemes

The task of an authentication scheme is to tell if a username/password combination is valid. By default Daisy uses its own authentication based on Daisy's user management. New authentication schemes can be added to check against external systems. Daisy ships with example authentication schemes for LDAP and NTLM which are usable through simple configuration. If you have other needs, you can implement your own scheme. See also user management (page 82) and an example (page 137).

An authentication scheme should implement the following interface:

org.outerj.daisy.authentication.spi.AuthenticationScheme[10]

4.16.1.2.4 Text extractors

Text extractors extract text from various content formats for the purpose of full text indexing. Daisy includes a variety of such text extractors, e.g. for MS Word and PDF.

A text extractor should implement the following interface:

org.outerj.daisy.textextraction.TextExtractor[11]

4.16.1.2.5 Link extractors

Link extractors extract Daisy document links from various content formats. These extracted links are maintained by the repository to enable searching which documents link to a certain document.

A link extractor should implement the following interface:

org.outerj.daisy.linkextraction.LinkExtractor[12]

4.16.1.2.6 HTTP handlers

Adding new functionality to the HTTP interface of the repository server can be done by implementing a request handler:

org.outerj.daisy.httpconnector.spi.RequestHandler[13]

This is mostly useful for extensions which want to support remote invocation.

4.16.1.2.7 Other

It's possible to add it any sort of component to be launched as part of the repository server, it doesn't necessarily need to register something in the plugin registry. Such components could perform all sorts of tasks such as listening to JMS or synchronous repository events, performing timed actions, etc.

4.16.1.3 Plugin registry

To make plugins available, you need to register them with a service called the PluginRegistry[14].

When registering a plugin, you need to specify the following:

- The plugin type interface, specified as a Java Class object
- A name for the plugin, which should be unique within a particular type of plugins
- The actual plugin instance (an object implementing the plugin type interface)

For example, a text extractor is registered like this:

```
import org.outerj.daisy.textextraction.TextExtractor[15];
...

TextExtractor extractor = new MyTextExtractor();

pluginRegistry.addPlugin(TextExtractor.class, "my text extractor", extractor);
```

To gain access to the PluginRegistry, use `daisy:import-service` to import it into your spring container:

```
<beans ...

  <daisy:import-service id="pluginRegistry"
service="org.outerj.daisy.plugin.PluginRegistry"/>
```

For a complete example, see the authentication scheme sample (page 137).

It is recommended to unregister the plugin when shutting down.

That's all you need to know about registering plugins. It is the responsibility of the PluginRegistry and the user of the plugins to handle the rest.

4.16.1.4 Deploying plugins

As explained earlier, a plugin should be packaged as a container jar.

The Daisy Runtime configuration for the repository server will automatically include container jars put in the following directories:

```
<daisy data dir>/plugins/load-before-repository

<daisy data dir>/plugins/load-after-repository
```

To see how these are included in the runtime configuration, have a look at `<DAISY_HOME >/repository-server/conf/runtime-config.xml`

As explained in the Daisy Runtime documentation, the container jars put in these directories will be loaded in alphabetical order.

The reason to have two directories, load-before-repository and load-after-repository, is as follows:

For some plugins, it is desirable to register them before the core repository is started because they modify the behavior of the repository. For example, take text extractors (which extract text

for the purpose of fulltext indexing). From the moment the repository is started, it can start doing work. For example it might receive a JMS event of an updated document and fulltext-index it. If we would only register text extractor plugins after the repository is started, there will be a (small) amount of time during which the repository server will try to index documents without these additional text extractors being available. Hence documents handled during this period would be treated differently.

Other plugins might be depended on the repository being available (in Daisy Runtime speak: they import services exported by the repository container) , and hence can only be loaded after the repository is available.

In general, plugins which modify the behaviour of the repository server (text extractors, pre-save hooks, authentication schemes, etc.) should be put in the load-before-repository directory.

After copying the new plugin(s) into the appropriate directory, you need to restart the repository server for the plugin to be loaded.

4.16.1.5 Repository Extensions

Repository Extensions are a particular type of plugins that add extra functionality to the repository. An Extension is usually related to the repository, e.g. because it needs access to information in the repository. The extension code has no special privileges, it simply makes use of the repository using the normal APIs and SPIs.

In fact, since an extension is simply an application which makes use of the repository via its API, one could wonder why there is a need for adding this code as an extension to the repository. The reasons are:

- the extension can be retrieved by API users using the getExtension() method shown below, both for the local and remote API implementations (if the extension provides a remote implementation). So API users have a common way to get access to extension functionality.
- the extension automatically has access to the repository object from which it is retrieved. Otherwise one would have to supply the repository object as an argument to extension functions: doSomething(repository, other arguments)

Extensions registered with the repository can be retrieved like this:

```
MyExtension myExtension = (MyExtension)repository.getExtension("name-of-the-extension");
```

whereby MyExtension is the interface of the extension.

Examples of extensions delivered with Daisy: the NavigationManager, the EmailSubscriptionManager, the Emailer, the Publisher and the DocumentTaskManager.

To register in an extension you should register an implementation of this interface:

org.outerj.daisy.repository.spi.ExtensionProvider[16]

and add it to the plugin registry (page 136).

4.16.1.6 Sample: custom authentication scheme

As an example of how to build a repository plugin, here we will look at how to implement an authentication scheme. Other plugins follow a similar approach.

For the purpose of this example, we'll create a FixedPasswordAuthenticationScheme, i.e. an authentication scheme which accepts just one fixed password regardless of the user.

Components to be deployed in the repository server need to be packaged as a *container jar*, this is a normal jar file containing at least one Spring bean container definition. This is described in detail in the Daisy Runtime documentation (page 141).

4.16.1.6.1 The container jar

The container jar we're going to build will have the following structure:

```
org
  foobar
    FixedPasswordAuthenticationScheme.class
DAISY-INF
  spring
    applicationContext.xml
```

So we only need to create two files.

4.16.1.6.2 Implement the necessary code

For a custom authentication scheme, we need to do two things:

- make an implementation of the interface AuthenticationScheme
- register this implementation with the PluginRegistry

We do this here with one class, shown below. Save this code in a file called FixedPasswordAuthenticationScheme.java

```
package org.foobar;

import org.outerj.daisy.authentication.AuthenticationScheme;
import org.outerj.daisy.authentication.AuthenticationException;
import org.outerj.daisy.plugin.PluginRegistry;
import org.outerj.daisy.repository.Credentials;
import org.outerj.daisy.repository.user.User;
import org.outerj.daisy.repository.user.UserManager;
import org.apache.avalon.framework.configuration.Configuration;

public class FixedPasswordAuthenticationScheme implements AuthenticationScheme {
    private String name;
    private String description;
    private String password;
    private PluginRegistry pluginRegistry;

    public FixedPasswordAuthenticationScheme(Configuration config,
            PluginRegistry pluginRegistry) throws Exception {
        password = config.getChild("password").getValue();
        name = config.getChild("name").getValue();
        description = config.getChild("description").getValue();
        this.pluginRegistry = pluginRegistry;
        pluginRegistry.addPlugin(AuthenticationScheme.class, name, this);
        System.out.println("Scheme " + name + " added!");
    }

    public void destroy() {
        pluginRegistry.removePlugin(AuthenticationScheme.class, name, this);
    }

    public String getName() {
        return name;
```

```
    }

    public String getDescription() {
        return description;
    }

    public boolean check(Credentials credentials) throws AuthenticationException {
        // this is the actual password check, very simple in this case
        return password.equals(credentials.getPassword());
    }

    public void clearCaches() {
        // we have nothing cached
    }

    public User createUser(Credentials crendentials, UserManager userManager) throws
AuthenticationException {
        // unsupported
        return null;
    }
}
```

This file can be compiled like this:

Use the method of your choice to compile the code (Ant, Maven, your IDE, ...). When using the command below, we hope you know enough about this that everything should be on one line. For Windows, replace $DAISY_HOME with %DAISY_HOME% and the colons with semicolons)

```
javac -classpath
          $DAISY_HOME/lib/daisy/jars/daisy-repository-api-<version>.jar:
          $DAISY_HOME/lib/daisy/jars/daisy-repository-server-spi-<version>.jar:
          $DAISY_HOME/lib/daisy/jars/daisy-pluginregistry-api-<version>jar:
          $DAISY_HOME/lib/avalon-framework/jars/avalon-framework-api-4.1.5.jar
      FixedPasswordAuthenticationScheme.java
```

4.16.1.6.3 Create a Spring bean container definition (applicationContext.xml)

The following is the Spring container definition.

```
<?xml version="1.0"?>
<beans              xmlns = "http://www.springframework.org/schema/beans"
               xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
              xmlns:daisy = "http://outerx.org/daisy/1.0#runtime-springext"
               xmlns:conf = "http://outerx.org/daisy/1.0#config-springext"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                            http://outerx.org/daisy/1.0#runtime-springext
                            http://daisycms.org/schemas/daisyruntime-springext.xsd
                            http://outerx.org/daisy/1.0#config-springext
                            http://daisycms.org/schemas/config-springext.xsd">

  <daisy:import-service id="configurationManager"
            service="org.outerj.daisy.configuration.ConfigurationManager"/>
  <daisy:import-service id="pluginRegistry"
            service="org.outerj.daisy.plugin.PluginRegistry"/>

  <bean id="foo" class="org.foobar.FixedPasswordAuthenticationScheme"
destroy-method="destroy">
    <constructor-arg>
      <conf:configuration group="foobar" name="fixedpwd-auth"
source="configurationManager">
        <conf:default xmlns="">
```

```
          <name>fixedpwd</name>
          <description>Fixed global password</description>
          <password>jamesbond</password>
        </conf:default>
      </conf:configuration>
    </constructor-arg>
    <constructor-arg ref="pluginRegistry"/>
  </bean>

</beans>
```

The special `<daisy:import>` elements are used to import services from other containers. You can assume these services will be available. See the Daisy Runtime (page 141) for details on how this works.

The component configuration system is also explained elsewhere (page 147), for now it suffices to know that the content of the conf:default element will be supplied as default configuration to the component. This configuration could be customized through the myconfig.xml file.

4.16.1.6.4 Deploy the new authentication scheme

Create the jar file (which is a normal zip file) containing the FixedPasswordAuthenticationScheme.class and applicationContext.xml in the directory structure as outlined earlier.

```
$ find -type f
./org/foobar/FixedPasswordAuthenticationScheme.class
./DAISY-INF/spring/applicationContext.xml

$ jar cvf fixedpwd-auth.jar *
```

Then copy the jar file to the directory `<repo data dir>/plugins/load-before-repository`.

Now stop and start the repository server. If everything goes well, the authentication scheme will be loaded and the following line will be printed (to standard out if you're using the wrapper, the wrapper log file)

```
Scheme fixedpwd added!
```

If you go to the Administration console in the Daisy Wiki and create or edit a user, you will be able to select the new authentication scheme.

4.16.1.6.5 Follow-up notes

When doing this for real, you would of course use proper logging (page 149) instead of System.out.println.

For those familiar with Spring, instead of constructor dependency injection, we could as well have used setter dependency injection.

If the implementation of the authentication scheme requires some code on the classpath, than this can be specified by adding a DAISY-INF/classloader.xml file to the container jar. See the Daisy Runtime documentation.

Happy hacking!

4.16.2 Daisy Runtime

4.16.2.1 What is it?

The Daisy Runtime is the platform upon which the repository server runs. Basically, it consists of a set of isolated Spring[17] bean containers, with some infrastructure for setting up classloaders and for sharing services between the Spring containers. It's really just a thin layer around Spring. You could think of it as a “Spring-OSGI-light”.

Some background on how we arrived at this can be found in this blog entry[18].

4.16.2.2 How it works

The Daisy Runtime takes two important parameters for booting up:

- a list of *container jars* to start
- the location of a Maven1-style repository. This is needed because the classpaths for the containers are defined as sets of artifacts (page 0). An artifact is identified, as in Maven, by the tripple {group id, artifact id, version}.

In this documentation, the words “artifact” and “jar” basically mean the same thing.

Each container jar contains a file describing the the required classpath, and one or more Spring container definitions. The Daisy Runtime will start a separate Spring container corresponding to each container jar.

The details of the classloader setup are described further on.

Each container can export and import services to/from a common service registry. How this is done is also explained further on.

So once more, summarized: the purpose of the Daisy Runtime is starting Spring containers, setting up classloaders for them, and allowing them to share selected services.

4.16.2.3 The Runtime CLI

There are few ways to start the runtime:

- programatically using the Daisy Runtime API
- using the CLI (command line interface)

For both cases, we also have a small launcher jar which has the advantage that you only need to add this launcher jar to your classpath, and it will then set up the classloader to boot the Daisy Runtime. In case of the programmatic access, you will of course need the jars for any API's you want to use in the current classloader.

4.16.2.4 The runtime config file

The runtime config file is an XML file listing the container jars to be started as part of the runtime.

Container jars can be specified in multiple ways

- as an artifact reference
- as a file path
- as a directory, from which all jars are loaded (in alphabetical order)

The following sample shows how to specify each of them in the configuration:

```
<?xml version="1.0"?>
<runtime>
  <containers>

    <artifact id="cont1" groupId="foo" artifactId="bar" version="2.1"/>

    <file id="cont2" path="/path/to/file.jar"/>

    <directory id="foo" path="/foo/bar"/>

  </containers>
</runtime>
```

The id attribute values need to be unique.

The containers are started in the order as listed here. The order can be important for export and import of services, as described further on.

In the file and directory paths, you can insert system property using `${...}` notation. For example:

```
<directory id="something" path="${user.home}${file.separator}/containers"/>
```

4.16.2.5 The container jar

A container jar is normal jar file containing a DAISY-INF directory with metadata for the Daisy Runtime.

The structure is as follows:

```
DAISY-INF
 + classloader.xml
 + spring
    + applicationContext.xml
    + ...
```

4.16.2.5.1 The DAISY-INF/spring directory

The spring directory can contain Spring container definitions in the form of XML files. There needs to be at least one. The files can have any name as long as it ends on `.xml`. Often `applicationContext.xml` is used.

These are standard Spring XML files, though we have some extension namespaces for the import and export of services, and for the configuration system.

As a template, to have all these namespaces declared, you might use the following:

```
<?xml version="1.0"?>
<beans              xmlns = "http://www.springframework.org/schema/beans"
                 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
              xmlns:daisy = "http://outerx.org/daisy/1.0#runtime-springext"
```

```
               xmlns:conf = "http://outerx.org/daisy/1.0#config-springext"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                            http://outerx.org/daisy/1.0#runtime-springext
                            http://daisycms.org/schemas/daisyruntime-springext.xsd
                            http://outerx.org/daisy/1.0#config-springext
                            http://daisycms.org/schemas/config-springext.xsd">

  <!-- Import and export sample syntax:
  <daisy:import-service id="" service="(interface FQN)"/>
  <daisy:export-service ref="" service="(interface FQN)"/>
  -->

  <!-- Insert bean definitions here -->
</beans>
```

4.16.2.5.2 The classloader.xml file

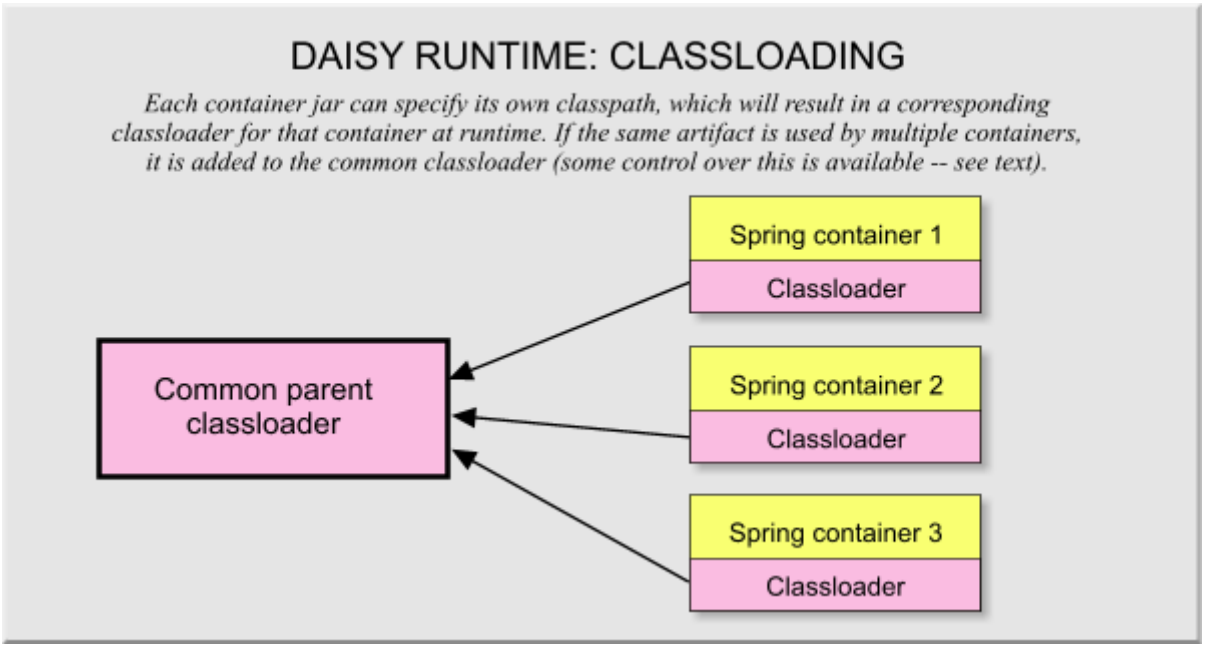
The classloader.xml file lists the artifacts that need to be in the classpath. The container jar itself is always automatically added as the first entry in the classpath. The classloader.xml file is optional.

As mentioned before, jars are referenced as Maven artifact references. The Daisy Runtime will search them in a Maven-style local repository specified at startup of the Runtime. The Runtime does not support automatic downloading of missing artifacts from remote repositories.

The syntax of the classloader.xml is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<classloader>
  <classpath>
    <artifact groupId="" artifactId="" version="" share="allowed|required|prohibited"/>
  </classpath>
</classloader>
```

This is pretty straightforward, except for the `share` attribute. Next to the classloader for each container, the Daisy Runtime also creates a common classloader that acts as the parent classloader for each of the container classloaders, as illustrated in the figure below.



By means of the share attribute, you can specify if an artifact can, should or should not be added to the common classloader. This is done using the following attribute values:

- `allowed`: the artifact that can be put in the common classloader, but doesn't need to. The system can add the artifact to the common classloader if multiple containers use the artifact, and they all use the same version. However, things should just as well work when they are in the container-specific classloader. Typically used for all sorts of (third party) library code.
- `required`: the artifact must be added to the common classloader. Typically used for the API of exported services and any classes referenced by those.
- `prohibited`: the artifact that should not be added to the common classloader, because it is private implementation specific to the current container. Typically only used for the specific implementation classes of the functionality provided by the container. These implementation classes are often in the container jar itself, which is never added to the common classloader, so the share-prohibited is not used very often.

When starting up, the Daisy Runtime will first read the classloader configurations of all containers in order to determine what artifacts should be part to the common classloader and what artifacts should be in container-specific classloaders. When using the Runtime CLI, you can specify the command line option `--classloader-log` to see a report of this.

4.16.2.5.2.1 Enforcement of share=”prohibited”

Currently share=”prohibited” is not enforced. For example if one container has artifact A as share-allowed and another one has artifact A as share-prohibited, then artifact A will be added to the common classloader, and the Daisy Runtime will only print a warning.

4.16.2.5.2.2 Disabling artifact sharing

Since the share=”allowed” mode only indicates optional sharing, things should work just as well when these artifacts are not shared. The Daisy Runtime CLI supports the command line option to do this, --disable-class-sharing.

This can be useful to check that the classpaths of all containers list their required dependencies, and that things which should be common use the share=”required” mode.

4.16.2.5.2.3 Sharing/publishing classes requires to put them in a different jar

As a consequence of the how our system works, if you want to add classes to the common classloader they have to be in a separate jar. It is good practice to put APIs and implementation in separate jars, so often this is no problem. This is different from e.g. OSGI where exporting is done using Java packages rather than jars.

4.16.2.5.2.4 Dependencies between artifacts

If the system decides to put one shareable jar (jar A) in the shared classloader, and another shareable jar (jar B) not, and jar A is dependent on jar B, there might be problems since the classes in jar A won't find the classes in jar B. However, this is an unlikely situation to occur since then all containers should have both jar A and B as shareable jars.

4.16.2.5.2.5 The order of the entries in the common classloader

There is currently no control over the order of the entries in the common classloader.

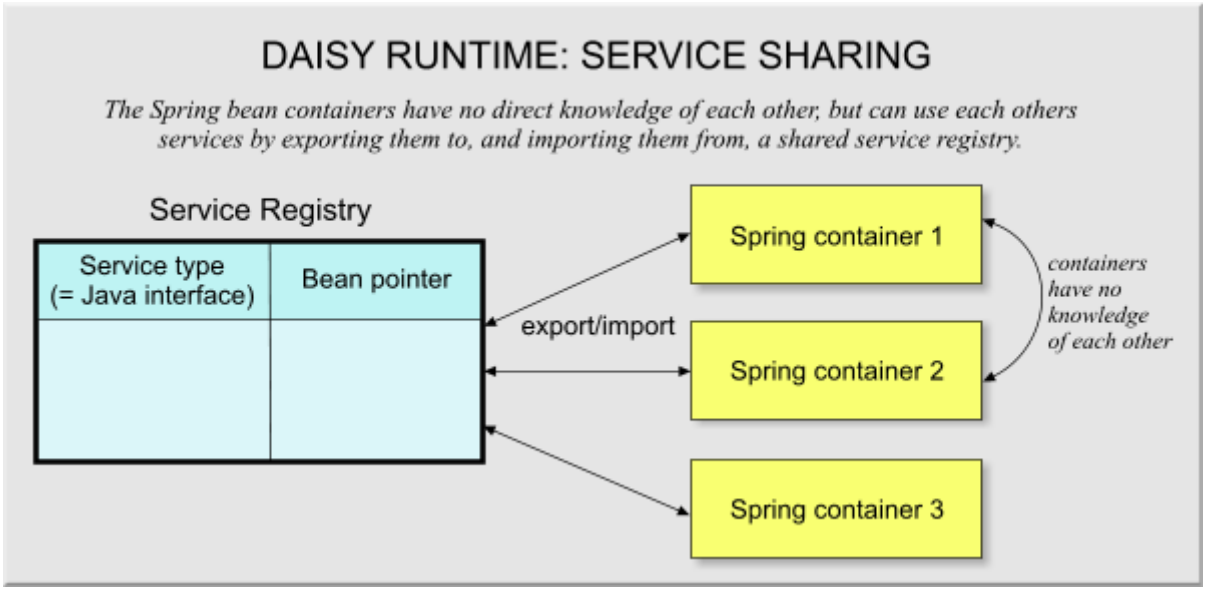
4.16.2.5.2.6 Knowing more about classloaders

For using the Daisy Runtime, it suffices to have a very basic understanding of classloaders. If you are interested on learning everything about classloading, a good book is Component Development for the Java Platform[19].

4.16.2.6 Exporting and importing services

By default one container cannot access the “beans” in another container. The Daisy Runtime provides the ability to export specific beans as “services” and to import services exported by other containers.

This importing and exporting is done by custom elements in the Spring XML container definition.



4.16.2.6.1 Exporting a bean / service

When exporting a service, you need to specify the ID of the bean and the interface which you want to make available:

```
<daisy:export-service ref="mybeanid" service="org.mydomain.MyInterface"/>
```

If a bean implements multiple interfaces, and you want to make this all available as services, you need multiple exports for the same bean:

```
<daisy:export-service ref="mybeanid" service="org.mydomain.InterfaceA"/>
<daisy:export-service ref="mybeanid" service="org.mydomain.InterfaceB"/>
```

Note that the service must be an interface, it is not possible to export services using concrete classes.

4.16.2.6.1.1 Only one service of a certain type can be shared

The shared service registry is basically a map using the service interface as key. This means there can be only service per interface. An exception will be thrown if a second export for the same service interface is done.

4.16.2.6.2 Importing a service

To import a service, use the following syntax:

```
<daisy:import-service id="datasource" service="javax.sql.DataSource"/>
```

The id is the id for the bean in the local container. You will hence be able to pass the service to other beans using this id (as you would do for any other bean).

4.16.2.6.2.1 Order in which the containers are loaded

The order in which the container jars are specified to the Daisy Runtime is important, it needs to be such that the imports of a container are satisfied by exports done by earlier started containers.

4.16.2.6.2.2 Shielding of exported services

Daisy will not provide a direct pointer to the corresponding bean when importing a service. Rather, it creates a dynamic proxy implementing the service interface. This is to:

- (main reason) only allow calling methods part of the service interface
- this also allows to switch the context classloader when calling the service
- and to check that the container containing the bean is still alive

4.16.2.7 Daisy Runtime shutdown

When shutting down the Daisy Runtime, all Spring containers are destroyed in the reverse order that they were started.

4.16.2.8 Starting the Daisy Runtime using the CLI

The Daisy Runtime CLI provides a couple of useful options, use the -h (help) argument to see them.

For example, in the binary distribution the Daisy Runtime is started by the daisy-repository-server script:

```
cd $DAISY_HOME/repository-server/bin
daisy-repository-server -h
```

In development setup, this is

```
cd <source tree root>/repository/server
start-repository -h
```

4.16.3 Component configuration

4.16.3.1 The API

The current configuration system is an interim solution, mainly for backwards compatibility with our older runtime environment (Avalon Merlin). Nonetheless, it's not bad, though in the future we'll likely add more advanced features like runtime reloading and notification of configuration changes, and drop the use of Avalon-specific APIs.

For representing configuration information, we use, for historical reasons, the Avalon Configuration API. In this API, the configuration data is modeled as a tree of Configuration[20] objects. Each Configuration object can have attributes and children. This structure maps quite well onto XML, though it doesn't support mixed content (= sibling text and element nodes). The Configuration API provides convenient methods for retrieving the data as various non-string types, e.g. Configuration.getAttributeAsInteger(name)

Components (beans) in need of configuration data don't read this directly from files, but retrieve it using a logical name from a component called the ConfigurationManager.

While beans could depend on the ConfigurationManager component, we rather not let them retrieve the information themselves, but supply them directly with the Configuration object. Also, sometimes we'd like to be able to specify default configuration information. For these purposes, we made a Spring extension.

4.16.3.1.1 Example

Suppose we have a class Foo like this:

```
import org.apache.avalon.framework.configuration.Configuration

public class Foo {
    public Foo(Configuration conf) {
       System.out.println(conf.getChild("message").getValue());
    }
}
```

We could now supply it with the Configuration object like this:

```
<beans              xmlns = "http://www.springframework.org/schema/beans"
             xmlns:daisy = "http://outerx.org/daisy/1.0#runtime-springext"
              xmlns:conf = "http://outerx.org/daisy/1.0#config-springext"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                            http://outerx.org/daisy/1.0#runtime-springext
                            http://daisycms.org/schemas/daisyruntime-springext.xsd
                            http://outerx.org/daisy/1.0#config-springext
                            http://daisycms.org/schemas/config-springext.xsd">

  <daisy:import-service id="configurationManager"
                        service="org.outerj.daisy.configuration.ConfigurationManager"/>

  <bean id="thing1" class="Foo">
    <constructor-arg>
      <conf:configuration group="mythings" name="foo" source="configurationManager"/>
    </constructor-arg>
  </bean>
```

```
</beans>
```

Note the following things:

- A namespace and schema location is defined for the configuration stuff
- We import the ConfigurationManager service, assuming it is exported by another container in the Daisy Runtime (which is the case for the repository server)
- We use the `<conf:configuration>` element to get the configuration from the ConfigurationManager. The group and name attribute are for addressing the configuration.

Optionally, one can specify default configuration:

```
  <bean id="thing1" class="Foo">
    <constructor-arg>
      <conf:configuration group="mythings" name="foo" source="configurationManager">
        <conf:default xmlns="">
          <message>No message configured!</message>
        </conf:default>
      </conf:configuration>
    </constructor-arg>
  </bean>
```

4.16.3.2 Configuring the configuration

So where does the ConfigurationManager gets its configuration from? From an XML file, for a default repository installation this file is called `myconfig.xml` and can be found in the repository data directory.

The myconfig.xml looks like this:

```
<targets>
  <target path="mythings/foo">
    <configuration>
      <message>Hello world!</message>
    </configuration>
  </target>
</targets>
```

The format of this file is again, for compatibility reasons, the same as it was for Avalon Merlin, where the path attributes pointed to targets (components) for which the configuration was intended.

The path attribute contains the group/name specified when retrieving the configuration. There can be as many of these `<target>` elements as desired.

4.16.3.2.1 Backwards compatible paths

If you look in the actual myconfig.xml of the repository server, you'll see things like this:

```
<target path="/daisy/repository/blobstore">
```

This path attribute does not correspond to the group/name structure. This path attribute is however backwards compatibility with the old runtime. Supporting these old paths avoids the

need for users to update their existing configuration files and (for us) to adjust utilities which read/update the configuration file.

The ConfigurationManager maintains a mapping of old paths to new paths, this mapping is used when reading the myconfig.xml.

4.16.3.3 Configuration merging

When a default configuration is specified (using conf:default), and there is also an actual configuration, both are merged by means of a CascadingConfiguration[21].

4.16.3.4 Further exploration

The sources of the configuration stuff can be found in the Daisy source tree at

```
services/configuration
```

4.16.4 Logging

4.16.4.1 Logging API

In the repository server all logging is performed using the commons-logging[22] API.

It is then up to the environment in which the repository server is started to set up a concrete logging implementation.

In the Daisy Runtime CLI we replaced the commons-logging API by its clone jcl-over-slf4j[23], the actual logging engine is log4j[24].

4.16.4.2 Logging tips

The Daisy Runtime CLI has some handy options for logging, mostly intended for use during Daisy development:

- `-l` (lowercase L) enables logging to the console, for the log level specified as argument (i.e. debug, info, warn, error)
- `-m` specifies the category for which to enable this logging (optional, by default it's for the root logger)

So for example:

```
start-repository -l debug
```

will cause all debug log to be sent to the console (during startup this will be quite a lot). To only see logging produced by Daisy, one could do:

```
start-repository -l debug -m org.outerj.daisy
```

4.16.5 Launcher

The purpose of the launcher is to easily start the repository server without the need to add all the required implementation jars to the classpath of your project. When using the launcher, you only

need the launcher jar on the classpath, and the launcher will then construct a classloader containing the required dependencies. This also means that you don't need to update classpaths if they change between Daisy versions.

More precisely, the launcher supports launching of 3 different things:

- the Daisy Runtime CLI
- the Daisy Runtime
- the remote repository client

Except for the CLI, you usually start these things with the purpose of being able to talk to them. For this, you still need the required repository APIs in your classloader.

The launcher jar is also executable using java -jar, in which case it will start the Daisy Runtime CLI.

Some pointers to examples:

- The Daisy Runtime CLI launcher is used by the daisy-repository-server script and repository-server service wrapper
- The Daisy Runtime and remote repository client launchers are used by the testcases found in the source tree below repository/test. If you'll look in the Maven pom there, you'll see it only has the various API jars, and the launcher jar, as dependencies. The code using the launcher is in AbstractDaisyTestCase.

4.17 Repository Implementation

This section contains some information on internals of the repository server. It is mostly only relevant for people who want to hack on Daisy itself.

4.17.1 Repository Implementation

We have mentioned before that there are two implementations of the repository API: one we call **local** (the one in the repository server) and one we call **remote**. In this document we're going to look into how the repository objects are implemented to support both local and remote implementations.

4.17.1.1 Repository server implementation

The implementation of the repository can be found in the source tree in the *repository* directory. This is the structure of the repository directory:

```
+ repository
    + api
    + client
    + common
    + server
    + server-spi
    + spi
    + test
    + xmlschema-bindings
```

The api directory contains the repository API definition, these are mainly interfaces. Repository client applications only need to depend on the api jar and the xmlschema-bindings jar (thus when compiling a client program, only these jars need to be in the classpath).

The client directory contains the remote implementation, the server directory contains the local implementation. The common directory contains classes common to both implementations (this is discussed further on). The spi directory contains "Service Provider Interfaces", these are interfaces towards extension components. The server-spi directory is similar to the spi directory, but contains interfaces that are only relevant for the server implementation. The test directory contains functional tests. These tests automate the process of constructing an empty database, starting an embedded repository server, and then execute a JUnit test. The xmlschema-bindings directory contains XML schemas for the XML formats used by the repository, these are compiled into corresponding Java classes by use of XMLBeans. Logically speaking these classes are a part of the repository API.

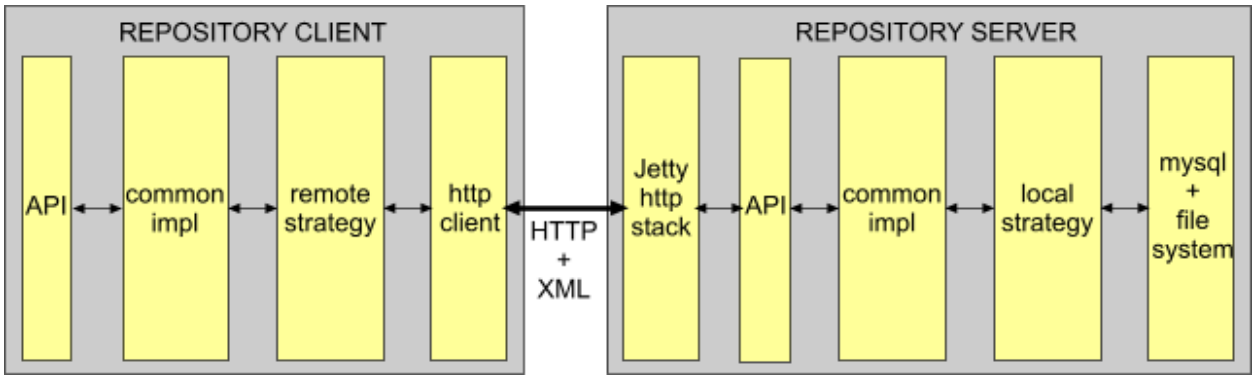
Next to the repository directory, the *services* directory also contains a good amount of functionality that is used by the repository client or server. The subprojects in the service directory are however separated from the main repository code because either they are completely independent from it (and reusable and testable outside of the Daisy repository code), or it are repository extensions.

4.17.1.2 The local, remote and common implementations

If we consider an entity such as a Document, a User or even an Acl, there's a lot of the implementation of these interfaces that will be equal in the local and remote interface: in both cases they need instance variables to hold the data, and implementations of the various methods. In fact, for most of these entities, the only difference is the implementation of the save method. In the local implementation, the save method should update the database, while in the remote implementation, the save method should use an appropriate HTTP call to perform the operation.

Therefore, the basic implementation of these objects is separated out in the "common-impl" subproject, which delegates things that are specific for local or remote implementation to a certain Strategy interface. The Strategy interface is then implemented differently for the local and remote implementations.

The diagram below depicts this basic organisation of the code.



Not all operations of the repository are of course loading and saving entities, there are also items such as querying and ACL-checking, and these are also delegated by the common-impl to the appropriate strategy instance.

So practically speaking, we could say that most of the real work happens in the implementations of the Strategy interfaces. You can find the Strategy interfaces by going to the repository/common directory and searching for all *Strategy.java files.

4.17.1.3 User-specific and common objects

Lets do a quick review of the Daisy API, for example:

```
// Getting a repository instance
Repository repository = repositoryManager.getRepository(new Credentials("user",
"password"));

// Retrieving a document
Document document = repository.getDocument(55, false);

// Creating a collection
CollectionManager collectionManager = repository.getCollectionManager();
DocumentCollection newCollection = collectionManager.createCollection("abc");
newCollection.save();
```

The RepositoryManager.getRepository() method returns a caller-specific Repository object. With "caller-specific", I mean a custom instance for the thread that called the getRepository() method. The Repository object remembers then the user it belongs too, so further methods don't need a credentials parameter.

If later on we do a call to repository.getCollectionManager(), the returned collectionManager instance is again caller-specific, thus it knows it represents the authenticated user and we don't need to specify credentials when calling further methods.

The implementations of interfaces like Repository (RepositoryImpl) and CollectionManager (CollectionManagerImpl) delegate internally the calls simply to a CommonRepository instance and a CommonCollectionManager instance, calling similar methods on them but with an additional user parameter. Thus RepositoryImpl and CollectionManagerImpl in a sense exist only to remember the authenticated user.

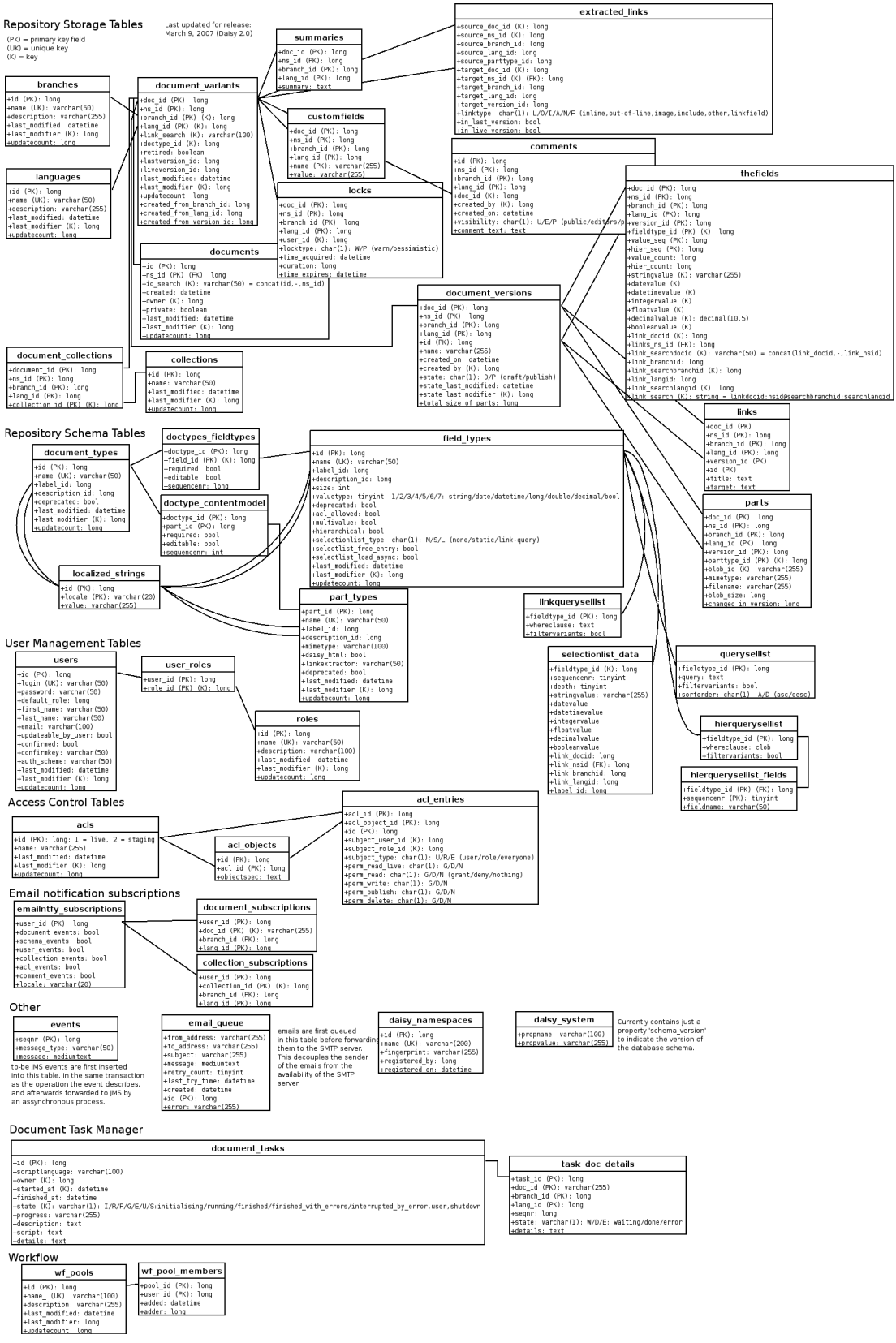
A diagram which illustrates all this is available here (page 0). (it's not embedded in the page because it is a bit wide).

As you'll see in the diagram, a call for getDocument on the repository delegates this call to the CommonRepository instance, which then in itself delegates the call to a LoadStoreStrategy class. If the CommonRepository class simply forwards the call to the LoadStoreStrategy class, you could wonder why this extra layer of delegation still exists and thus why the CommonRepository and its corresponding LoadStoreStrategy are not merged into one. Or in general, why this is not done for all *Manager classes (CollectionManager, AccessManager, UserManager, etc.) and their corresponding Common* classes. For a big part, this is because this is how it historically evolved, though the Common* classes still perform some functions such as caching which are (sometimes) shared between the local and remote implementations.

4.17.2 Database schema

The image below shows the database schema of the daisy repository. The actual content of parts is stored in files on the hard disk, the blob_id column in the parts table contains the filename (or more correctly, the id used by the BlobStore component to retrieve the data, but this is currently the same as the file name).

Never make changes to the database directly, always use the repository APIs.



Notes

1. javadoc:org.outerj.daisy.repository.VariantKey
2. javadoc:org.outerj.daisy.repository.HierarchyPath

3. http://jakarta.apache.org/lucene/
4. http://jakarta.apache.org/lucene/docs/queryparsersyntax.html
5. http://jakarta.apache.org/lucene
6. javadoc:root
7. javadoc:root
8. http://curl.haxx.se/
9. javadoc:org.outerj.daisy.repository.spi.local.PreSaveHook
10. javadoc:org.outerj.daisy.authentication.spi.AuthenticationScheme
11. javadoc:org.outerj.daisy.textextraction.TextExtractor
12. javadoc:org.outerj.daisy.linkextraction.LinkExtractor
13. javadoc:org.outerj.daisy.httpconnector.spi.RequestHandler
14. javadoc:org.outerj.daisy.plugin.PluginRegistry
15. javadoc:org.outerj.daisy.textextraction.TextExtractor
16. javadoc:org.outerj.daisy.repository.spi.ExtensionProvider
17. http://www.springframework.org
18. http://brunodumon.wordpress.com/2007/07/16/daisy-runtime/
19. http://www.amazon.com/Component-Development-Platform-Stuart-Halloway/dp/0201753065
20. http://excalibur.apache.org/apidocs/org/apache/avalon/framework/configuration/Configuration.html
21. http://svn.apache.org/viewvc/avalon/tags/CONFIGURATION_1_1_RC2/avalon-excalibur/configuration/src/java/org/apache/excalibur/configuration/CascadingConfiguration.java?revision=17791&view=markup
22. http://jakarta.apache.org/commons/logging/
23. http://www.slf4j.org/
24. http://logging.apache.org/log4j/

5 Daisy Wiki

The Daisy Wiki is a generic web-based frontend to the repository server. It provides both publishing and editing/management features. Please see the feature overview page (page 0) for a comprehensive introduction.

5.1 Daisy Wiki Sites

5.1.1 What is a Daisy Wiki "site"?

A Daisy Wiki site is a specific view on a Daisy Repository. A site is configured with a *default collection* (the concept of document collections is explained on the documents (page 46) page). Full text searches and recent changes are automatically limited to only show documents from that default collection. New documents created via the site are by default assigned to that collection. Each site can have its own navigation tree, and is configured with a specific document as the homepage of the site.

A site is configured with a certain branch and language: for any document consulted via that site, the shown document variant will depend on that branch and language.

The Repository Server isn't aware of the concept of sites, nor does the site concept partition the repository in any way.

5.1.2 Defining sites

Sites are defined by creating a directory for the site and putting a siteconf.xml file in it. This directory should be created in the "sites" directory. By default, this sites directory is located at:

```
<wikidata directory>/sites
```

The location of this directory can be changed in the cocoon.xconf.

The content of the siteconf.xml file should strictly adhere to a certain schema (thus no extra elements/attributes are allowed), otherwise the site will be ignored (in that case, an error will be logged in Cocoon's log files).

5.1.2.1 siteconf.xml syntax

An example siteconf.xml is displayed below.

```
<siteconf xmlns="http://outerx.org/daisy/1.0#siteconf">
  <title>foobar</title>
  <description>The "foobar" site</description>
  <skin>default</skin>
  <navigationDocId>1-DSY</navigationDocId>
  <homepageDocId>2-DSY</homepageDocId>
  <!-- homepage>....</homepage -->
  <collectionId>1</collectionId>
  <!-- collectionName>myCollection</collectionName -->
  <contextualizedTree>false</contextualizedTree>
  <!-- navigationDepth>4</navigationDepth -->
  <branch>main</branch>
  <language>default</language>
  <defaultDocumentType>SimpleDocument</defaultDocumentType>
  <publisherRequestSet>default</publisherRequestSet>
  <siteSwitching mode="all"/>
  <newVersionStateDefault>publish</newVersionStateDefault>
  <locking>
    <automatic lockType='pessimistic' defaultTime='15' autoExtend='true'/>
  </locking>
  <!--
  <documentTypeFilter>
    <include name="foo*"/>
    <exclude name="bar*"/>
  </documentTypeFilter>
  -->
</siteconf>
```

Element	Required	Description
title	yes	a (typically short) title for the site
description	yes	a description for the site, shown on the sites overview page
skin	yes	the skin to use for this site
navigationDocId	yes	the ID of the navigation document
homepageDocId	one of these	the ID of the homepage
homepage		a path to the homepage, used instead of the homepageDocId. Usually this is a path to a Wiki extension (`ext/something`)
collectionId	one of these	the ID of the default collection for the site
collectionName		the name of the default collection of the site. Will only be used when collectionId is not set.
contextualizedTree	yes	true or false. Indicates whether the navigation tree should be shown in full (= when false), or if the navigation tree should only have open branches leading to the selected node (= when true)
navigationDepth		always displays the first n levels of the navigation tree. When using this with contextualizedTree=true then the first n levels will always be shown no matter what and more may be shown as you progress through the navigation. When using contrextualizedTree=false then only the first n levels will be shown no matter at what place the current document happens to be in the navigation.

branch	no, default main	default branch for the site (specify either the branch ID or name)
language	no, default "default"	default language for the site (specify either the language ID or name)
defaultDocumentType	no	the default document type for this site. The document type can be specified either by ID or by name.
publisherRequestSet	no	which publisher request set to be used for the p:preparedDocuments publisher request (page 103) for pages rendered in this site.
siteSwitching	no	defines if the browser should be redirected to another site if a document is better suited for display in another site. Valid values for the mode attribute are: stay (never switch to another site), all (consider all available sites as sites to switch to), selected (consider only selected sites, listed in <site > child elements inside the <siteSwitching> element). For more information see URL space management (page 177).
newVersionStateDefault	yes	publish or draft. This indicates the default state of the "Publish changes immediately" flag on the edit screen.
locking		the locking strategy to use. To use no locking at all, remove the <automatic> element (but leave the empty <locking> element). To use warn-locking, i.e. only warning that someone else is editing the page but still allowing concurrent edits, change the lockType attribute to "warn".
documentTypeFilter	no	allows to specify a filter for the document types that should be visible when creating a new document within this site. Zero or more include and/or exclude patterns can be specified, the order of the patterns is of no importance. Document types will be shown if they match at least one include pattern and no exclude pattern. If there are only exclude patterns, an implicit `<include name="*"/>` is assumed. The patterns can be literal strings, or can contain the wildcards * and ?. The wildcard * matches zero or more characters. The wildcard ? matches exactly one character. To match one or more characters, you can use ?*. While document type names can't contain these characters, for completes we mention that the wildcards can be escaped using * and \?, and backslash when used in this context can be escaped using \\ (thus * and \\?).

		Note that this feature is not an access control, it forbids nothing, it just filters the document type list when shown.

5.1.2.2 Creating a site

Again, all you need to do to define a new site is creating a new subdirectory in the sites directory and putting a valid siteconf.xml in it.

5.1.2.3 Removing a site

To make a site unavailable, you can:

- remove its directory from the sites directory
- rename its siteconf.xml to something else
- make its siteconf.xml invalid (not recommended, just mentioning this for completeness as a reason a site may not appear)

5.1.2.4 Runtime detection of new/updated/deleted siteconf's

Changes to the sites configurations are automatically picked up, it is not needed to restart the Daisy Wiki. It can take up to 10 seconds before Daisy notices your changes (this interval is configurable in the cocoon.xconf). If you don't see a site appearing, check the cocoon log files for errors.

5.1.2.5 Site filtering

The list of sites displayed to the user is filtered based on whether the user has access to the homepage document of the site. In case a custom homepage path is used (<homepage> instead of <homepageDocId>), you can still specify the homepageDocId to cause filtering. If this is not done, the site will always be displayed in the list.

5.1.3 Creating a new site using daisy-wiki-add-site

If you want to create a new site, including a new collection, a new navigation tree and a new homepage document, you can use the daisy-wiki-add-site program for this, which will automatically perform these steps for you and put a new siteconf.xml in the sites directory. To do this, open a command prompt, make sure DAISY_HOME is set, go to DAISY_HOME/install and execute:

```
daisy-wiki-add-site <location of wikidata directory>
```

5.1.4 Other site-features

5.1.4.1 skinconf.xml

Maintaining a custom skin can be more work then you'd like to put into it, therefore it is also possible to customise (or parameterise) existing skins. For example, for the default skin you can alter the logo in this way.

This is done by putting a file called skinconf.xml in the appropriate site directory. The contents of this file will be merged in the XML pipelines and hence be available to the XSL stylesheets. The required content and format of this file depends upon what the skin you use expects.

If a site-specific skinconf.xml is not provided, the system will use the skinconf.xml found in the root of the sites directory, if it exists.

For more information on skinconf.xml, see here (page 189).

5.1.4.2 Extension sitemaps

See Daisy Wiki Extensions (page 198).

5.2 Daisy Wiki Editor Usage Notes

5.2.1 Introduction

This document describes the editor used to modify pages stored in the document repository. The editor features wysiwyg editing.

5.2.1.1 Where do I find the editor?

The editor can be reached by either editing an existing document or creating a new document.

To edit an existing document, use the Edit link in the document action menu. This link is only visible if you are allowed to edit the document.

To create a new document, select the New Document link in the menu. You are then first presented with a list of document types, after selecting one the editor will open.

5.2.1.2 Document type influence

The content of the edit screen depends somewhat on the document type of the document you're editing or creating. See documents (page 46) for a general discussion on documents and document types. As a quick reminder, a document can consists of multiple parts and fields. The parts contain the actual content, the fields are for more structured (meta)data.

If a part is marked as a "Daisy HTML" part, you will be presented with a wysiwyg editor for that part. Otherwise, a file upload control will be shown. Because of the ability to plugin in custom part editors (page 214), other types of editors might also appear (e.g. for editing book definitions).

5.2.1.3 Supported browsers

In theory, the document editing screen should work on most browsers. However, to use wysiwyg editing, it is advisable to use a recent version of one of the mainstream browsers, Mozilla/Firefox or Internet Explorer. We do most of our testing using recent versions of Firefox and Internet Explorer 6/7.

On other browsers, the editor will fall back to a textarea allowing you to edit the HTML source. On browsers that support wysiwyg editing, you can also switch to source editing.

In any case, Javascript (and cookies) must be enabled.

5.2.1.4 Heartbeat

While editing a page, the server keeps some state about your editing session. After a certain period of inactivity, the server will clean up the editing session. To avoid the editing session to expire while you're working on a document, a 'heartbeat' signal keeps your session alive. The heartbeat signal also serves to extend your lock on the document. (Technically speaking, the heartbeat signal is an Ajax-request).

5.2.1.5 Document locking

When you edit an existing document, the daisywiki will automatically take an exclusive lock on the document to ensure nobody else can edit the document while you're working on it. The duration of the initial lock is 15 minutes, the lock is then automatically extended if needed via the heartbeat signal.

If you start editing a page but decide you didn't want to after all, it is best to use the "Cancel editing" button at the bottom of the edit screen, so that your lock get cleared. If you don't do this, the lock will expire after at most 15 minutes, so this is not a big problem.

The locking behaviour can be adjusted by the site administrator. For example, the locking can be turned of completely. However, we expect that in most cases it will be left to the default behaviour described here.

5.2.1.6 Editing multiple documents at once

Editing multiple documents concurrently in different browser windows or tabs is supported.

5.2.2 Supported HTML subset and HTML cleaning

Although a wysiwyg editor is shown for the "Daisy HTML" parts, the goal is to limit the editing to a subset of HTML mainly focussing on structural aspects of HTML. So forget fonts, colors, special styling tricks, embedded javascript, and so on. Inserting those while editing in source view won't work either, as the HTML is cleaned up on the server side.

This cleanup process can also be triggered manually, by pressing the "Cleanup edited HTML" button. This can be useful if you pasted content copied from an external application and you want to see how it will look finally. When switching from wysiwyg to source view, the cleanup is also performed.

5.2.2.1 Supported HTML subset

These are the supported tags (or "elements") and attributes:

- <html> and <body>
- <p> with optional attributes align and class. The class attribute is only kept if it has one of the following values: note, warn, fixme
-
- <pre> with optional class attribute. The class attribute is only kept if has one of the following values: query, include, query-and-include
- <h1>, <h2>, <h3>, <h4>, <h5>

- <a> with required attribute href. If the href attribute is missing, the <a> will be dropped.
- , , <sup>, <sub>, <tt>, <del>
- , <ol>,
- <blockquote>
- <img> with attributes src and align (optional)
- <table> with optional attribute class, <tbody>, <tr>, <td>, <th>. <td> and <th> can have the attributes colspan, rowspan and valign

All tags not listed above will be removed (but their character content will remain). On the block-type elements and images, the id attribute is supported. For the most accurate list of elements and attributes, have a look at the htmlcleaner.xml file (see below).

The supported tags can have any content model as allowed by the HTML DTD, but of course limited to the supported tags. If an element occurs in a location where it is not supported, an ancestor is searched where it is allowed and the containing element(s) are ended, the element inserted, and the containing elements reopened. This happens for example when a <table> occurs inside a <p>.

<b> and <i> are translated to and respectively, as are tags with font-weight/font-style specifications.

If two or more
 tags appear after one another, this is translated to a paragraph split. The meaningless
's that the Mozilla editor tends to leave everywhere are removed. Text that appears directly in the <body> is wrapped inside <p> elements.

 tags inside <pre> are translated to newlines characters.

The result is serialized as a XML-well-formed HTML document (not XHTML) (UTF-8 encoded). Lines are split at 80 characters (if possible), meaningless whitespace is removed.

All this should also ensure that the resulting HTML is (mostly) the same whether it is edited using Mozilla or Internet Explorer.

The supported tags, attributes and classes for <p> are not hardcoded but can be configured in a file (htmlcleaner.xml). However, making arbitrary adjustments to this file is not supported (the html-cleaner code expects certain tags to be there). Adding new tags or attributes should generally not be a problem, but those won't have the necessary GUI editing support unless you implement that also.

5.2.3 Images

Images can be inserted either by browsing for an existing image in the repository, or by uploading a new image in the repository. You can also insert images that are not in the repository, but available at some URL.

You can change the alignment of the images (using the usual text-align buttons), and change how the text flows around the image. This last option won't have effect in the PDF output.

Note that images are also documents in the repository, thus are versioned and such. If you have an updated version of an image you want to insert, it is recommend to NOT delete the existing image and upload the new image, but rather go to the document editor for that image (you can use the "Open image in new window" toolbar button for this), and upload the new version over there.

Currently it is hardcoded that images should have an "ImageData" part. They can however be of any document type.

5.2.4 Links

The format of links to other documents in the daisy repository is:

```
daisy:<document id>
for example:
daisy:167
```

The daisy link can furthermore include branch, language and version specifications:

```
daisy:<document id>@<branch name or id>:<language name or id>:<version id>
```

Each of these additional parts is optional. For example to link to version 5 of document 167, on the same branch and language as the current document variant, use:

```
daisy:167@::5
```

The `<version id>` can be a number or the strings `last` or `live`.

If you don't know the id of a document by heart (which is likely the case), use the "Create link by searching" button on the toolbar.

A link can furthermore contain a fragment identifier. A fragment identifier is used to directly link to a specific element (e.g. a heading or a table) in a document. For this you first need to assign an ID to the element you want to link to (there is an "Edit ID" for this on the toolbar), and then you can adjust the link. The link editor dialogs make it easy by allowing to browse for available element IDs.

5.2.5 Upload and link ("attachment")

TODO

5.2.6 Includes

5.2.6.1 Including other Daisy documents

It is possible to include other documents into the document you are editing. This can be done in two ways:

- using the toolbar button "Insert new include"
- (manual) choose "Include" in the style dropdown, and enter a "daisy:" link in the paragraph.

To look up the actual document to include, use the toolbar button "Search for document to include". This will automatically insert an appropriate "daisy:" link.

After the "daisy:" link, you can put some whitespace (e.g. a a space character), and then put whatever additional text you want. This text will not be published, but is useful to leave a comment (e.g. the name of the included document).

By default the included document will look exactly as when it is displayed stand-alone. It is however possible to let the headings in the document shift (e.g. let a h1 become a h2 or h3) so that it better fits within the context. This heading shifting can be specified by opening the "Include settings" dialog using a toolbar button. In the HTML source, the include preference is stored in an attribute named daisy-shift-headings.

The editor will automatically show a preview of the included documents. The preview shows the content of Daisy-HTML parts and the fields, however without document type styling or heading-shifting applied. There are toolbar buttons to refresh or remove the previews. The include previews are not editable, when you try to edit them you will be asked to open the included document in a new window. Due to limitted control over the in-browser editors, you might find ways to edit the include previews anyway (e.g. using drag and drop), however these edits will be ignored.

When a document is published (= displayed), the includes are processed recursively. If an endless include-recursion is detected, an error notice will be shown at the location of the include.

5.2.6.2 Including content retrieved from arbitrary URLs

It is also possible to include other sources into your document, for example "http:" or "cocoon:" URLs (however, see Include Permissions (page 315)). In that case, those URLs must produce an embeddable chunk of HTML in the form of well-formed XML. These includes are currently only supported in the HTML publishing, thus not for PDFs.

5.2.7 Embedded queries

It is possible to embed a query in a page. To do this, put your cursor on an empty line, and choose "Query" in the style dropdown. The style of the paragraph will change to indicate it will now be interpreted as a query. Then enter the query in the paragraph.

The query must be written in the Daisy Query Language (page 63). It is advisable to first try out your query via the "Query Search" page, and once it works and gives the expected results, to copy and paste it in the document.

If you save a document containing an invalid query, an error notice will be shown at the location of the query.

5.2.8 Query and Include

The "Query-Include" option allows to specify a query, and the documents returned by that query will be included (rather then showing the query results). This allows to quickly created an aggregated document without needing to manually insert includes.

It is not important what you put in the "select" part of the query, you can simply do "select id where ....".

In the same way as for includes, it is possible to specify that heading-shifting to be performed.

5.2.9 IDs and fragment identifiers

It is not only possible to link to a document, but also to a specific location in a document. The element to which you want to link (a header, image, ...) must have an ID assigned. To do this, place the cursor inside the element to which to assign the ID, and then press the "Edit ID" button on the toolbar.

Then to link to the specific element, just insert a link like you always do. Both the "Create link" and "Create link by searching" dialogs allow to select the ID from the target document (in the "fragment ID" field). In the HTML source, the target ID is specified in the link as in this example:

```
daisy:5#notes
```

This link will cause the browser to scroll to the element with an ID attribute with the value "notes". The part starting from the hash sign is called the "fragment identifier".

Explicit anchor elements (e.g. HTML `<a name="notes"/>`) are not supported, as these sort of elements are not visible in the wysiwyg editor and thus users would work blindly if these were used (deleting or moving them without being aware of it, and being impossible to edit in wysiwyg mode).

5.2.10 Editor shortcuts

Shortcut	Function
ctrl+b	bold
ctrl+i	italic
ctrl+z	undo
ctrl+y	redo
ctrl+c	copy
ctrl+x	cut
ctrl+v	paste
ctrl+1, ctrl+2, ...	switch to header level 1, 2, ...
ctrl+a	select all
ctrl+q	switch between bullets / no bullets
ctrl+r	remove formatting (same as the gum icon) (since Daisy 1.1)

5.2.11 Editing hints

5.2.11.1 Firefox and Mozilla

Pressing enter once in Firefox inserts a newline (a
). To start a new paragraph, press enter twice.

The toolbar buttons for cut/copy/paste won't work because of security restrictions, though you can configure Firefox to allow this for a specific site. More information is given when you click on one of these buttons while in Firefox. However, using the keyboard shortcuts you can perform these operations without any special configuration.

When you add a link or apply a styling to some words on the end of a line, it might be difficult (read: impossible) to 'move after' the link or styling. You can interrupt the link or styling by moving the cursor to the end of the line, and pressing the 'Remove link' or 'Remove formatting' button (thus without making a selection).

5.2.11.2 Internet Explorer (IE)

Merging table cells in IE works a bit counter-intuitive. You cannot simply select multiple cells and click on the merge cell button. Instead, put the cursor in one cell, and click on the merge cell button. You will then be asked how many rows and columns you want to merge.

5.2.11.3 All browsers

To copy content from one document to the other, it is recommended to open the source document in the editor and copy from there. This way you make sure you copy the original source content, e.g. with "daisy:" links intact.

5.2.11.4 Editing fields

5.2.11.4.1 Editing hierarchical field values

Hierarchical field values are entered as a slash-separated sequence. For example: `abc / def / ghi`. The whitespace around the separator slashes is not significant, it will be dropped. An extra slash at the start or end is allowed and will be ignored. Slashes separated by only whitespace will be dropped (i.o.w. they will not cause the addition of an element in the hierarchical path). If you want to use the slash character in a value itself, this can be done by escaping it as a double slash (//).

5.2.12 Character Set Information

By default, daisy is configured to use unicode (UTF-8) everywhere. For the part content you enter in the wysiwyg or source editor, you can use whatever unicode-supported characters (more correctly, it is limited by as far as Java supports unicode). Metadata however, such as the document name, fields, etc is stored in a relational database, MySQL, which needs to be configured with a certain encoding (in West-Europe often latin1) and hence is limited to the characters supported by that encoding. Contact your system administrator if you which to know what encoding that is, and thus to what characters (glyphs) you're limited.

5.3 Embedding multimedia and literal HTML

5.3.1 Introduction

Daisy includes some default document types for easily embedding multi media and literal HTML. There are no special tricks involved in their implementation, you could easily create them yourself, but they are included for convenience.

5.3.2 Embedding multi media

This explains how you can upload a flash animation, a movie or a sound fragment using the MultiMediaObject document type.

5.3.2.1 Usage

Create a new document, choose the document type MultiMediaObject, and upload the item. There are some fields available to control various options, like height, width and looping.

Then, in the document you want to embed the multimedia item, do an include of this multi media document. For this, use the "Insert new include" button on the toolbar (of the rich text editor), and enter the ID of the document to include (you can look it up with the "Search for document to include" button).

5.3.2.2 Implementation note

The MultiMediaObject document type is simply a regular document type with which a document type specific XSLT is associated which inserts the HTML `<object>` and `<embed>` tags.

5.3.3 Embedding literal HTML

The default "Daisy-HTML" parts only allow a small, structured subset of HTML. Sometimes you might want to enter whatever HTML you like, most often to create HTML-based multi media. Another common example is including content from third-party sites such as YouTube. In that case, you can use the "Literal HTML" document type.

5.3.3.1 Usage

Create a document of type "Literal HTML", and enter the HTML in the editor. The HTML should be well-formed XML and enclosed by `<html>` and `<body>` tags. If this is not the case, the editor will automatically clean the HTML up (there's a 'sponge' icon to trigger this cleanup).

Only the content of the `<body>` tag will remain when the document is published.

To embed the newly created literal HTML document into another document, use the normal document include functionality.

5.3.3.2 Publisher request note

If you are using custom publisher requests, be aware that you need to enable the inlining of the "LiteralHtmlData" part. You can add this to the default publisher request (usually called default.xml), as shown here:

```
...
  <p:prepareDocument inlineParts="LiteralHtmlData"/>
...
```

5.4 Navigation

5.4.1 Overview

Daisy allows to create hierarchical navigation trees for your site. Some of the features and possibilities:

- navigation trees are dynamically generated for the current user and the current document. Documents for which the user has no "read live" access are removed from the navigation tree, as are retired documents or documents which don't have a live version.
- a navigation tree is defined as an XML document, and the resulting navigation tree output is also an XML document (in the Daisy Wiki styled through XSLT).

- a navigation tree can be requested in 'full' or 'contextualized', in case of this last option the navigation tree only contains expanded branches for the nodes leading to the current document. In the Daisy Wiki, this option can be configured in the siteconf.xml file.
- a navigation tree can contain queries for automatic insertion of nodes.
- navigation trees are stored as normal documents in the repository. They should use the predefined "Navigation" document type. Versioning thus also works for the navigation tree source, the live version of the navigation tree is the one that will actually be used (but it can also be the last version, see also further on). Normal access control can be used to restrict who can edit the navigation tree or read it (it = the navigation tree source). Other then this, navigation trees are publicly accessible: everyone can request the navigation tree output of a given navigation document.
- a navigation tree can include other navigation trees. This allows to separate the management of a navigation tree over multiple (groups of) users, each one having the right to edit their part of the navigation tree. This also enables reuse of navigation trees in different locations.
- to view the same documents in combination with different navigation trees (e.g. for different target audiences), you can create multiple sites in the Daisy Wiki (page 155).
- a navigation tree can be generated in either live (the default) or last version mode: when in last mode, the last version of the navigation tree document will be used, as well as of any imported navigation tree. Queries embedded in the navigation tree will be executed with the "search_last_version" option, and documents which have no live version will also become visible. For document related nodes, the document name will be taken from the last version (if no node label is specified), and "read" permission will be required instead of just "read live". All this together is useful when working in a staging view (page 219).

The Daisy Wiki has an advanced GUI for editing the navigation trees, so that users are not confronted with the raw XML. It is of course possible to switch to a source view. Editing a navigation tree is done in the same way as any other document is edited.

It is possible to create readable URLs (i.e. URLs containing readable names instead of numbers) by basing the URL space on the navigation tree and assigning meaningful node IDs to nodes in the navigation tree. See the document about URL management (page 177).

The 'root' navigation document of a site is accessible through the [Edit navigation] link below the navigation tree, which is visible is you are logged on as a non-guest-role user. You can also get an overview of all navigation documents using this query:

```
select id, branch, language, name where documentType = 'Navigation'
```

5.4.2 Description of the navigation XML format

5.4.2.1 The empty navigation tree

The simplest possible navigation tree description is the empty one:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
</d:navigationTree>
```

5.4.2.2 Document node

Adding document nodes to it is easy:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:doc id="26"/>
  <d:doc id="32">
    <d:doc id="15"/>
  </d:doc>
</d:navigationTree>
```

As shown, the nodes can be nested.

By default , the navigation tree will display the name of the document as the label of a node. However, sometimes you might want to change that, for example if the name is too long. Also, when editing the navigation tree description as a source document, it will quickly become difficult to figure out what node stands for what. Therefore, you can add an attribute called "label" to the d:doc elements:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:doc id="26" label="Introduction"/>
  <d:doc id="32" label="Hot Stuff">
    <d:doc id="15" label="Fire"/>
  </d:doc>
</d:navigationTree>
```

By default the ID of a document node is the document ID, but you can assign a custom ID by specifying it in an attribute called nodeId. The custom ID should not start with a digit and not contain whitespace.

To link to a document on another branch or in another language, add a branch and/or language attribute on the d:doc element. The value of the attribute can be a branch/language name or ID. By default, documents are assumed to be on the same branch and in the same language as the navigation tree document itself.

5.4.2.2.1 Visibility

The d:doc node supports a visibility attribute. By default, nodes are visible. The visibility attribute allows to specify that a node should only become visible when it is active or should never be visible at all. Non-visible nodes are useful to have the navigation tree opened up to a certain point, without displaying a too-deeply nested hierarchy or a large amount of sibling nodes. In the Daisy Wiki, where the navigation tree controls the URL space, this can additionally be useful to control the URL path.

The syntax is:

```
<d:doc id="..." visibility="always|hidden|when-active"/>
```

5.4.2.3 Link node

To insert a link to an external location (a non-Daisy document), use the link element:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:doc id="26" label="Introduction"/>
  <d:doc id="32" label="Hot Stuff">
    <d:doc id="15" label="Fire"/>
  </d:doc>
  <d:link url="http://outerthought.org" label="Outerthought"/>
</d:navigationTree>
```

The attributes url and label are both required. The link element supports an optional id attribute.

5.4.2.3.1 Visibility

It is possible to hide link nodes depending on whether the user has read access to some other document (a guarding document). The syntax for specifying this document is:

```
<d:link url="http://www.daisycms.org" label="Daisy CMS"
        inheritAclDocId="78-DSY" inheritAclBranch="main" inheritAclLanguage="default"/>
```

As usual, specifying the branch and language is optional.

For the inheritAclDocId, one can use the special value "this" to refer to the Daisy document in which the navigation tree itself is stored. In this case, the value of the inheritAclBranch and inheritAclLanguage attributes, if present, is not used.

5.4.2.4 Group node

If you want to group a number of items below a common title, use the group element. The group element can optionally have an attribute called id to specify a custom id for the node (otherwise, the id is automatically generated, something like g1, g2, etc).

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:group label="Some title">
    <d:doc id="26" label="Introduction"/>
    <d:doc id="32" label="Hot Stuff">
      <d:doc id="15" label="Fire"/>
    </d:doc>
  </d:group>
  <d:link url="http://outerthought.org" label="Outerthought"/>
</d:navigationTree>
```

The group node also supports the visibility attribute, see the document node for more information on this.

5.4.2.5 Import node

To import another navigation tree, use the import element:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:group label="Some title">
    <d:doc id="26" label="Introduction"/>
    <d:doc id="32" label="Hot Stuff">
      <d:doc id="15" label="Fire"/>
    </d:doc>
    <d:import docId="81"/>
  </d:group>
  <d:link url="http://outerthought.org" label="Outerthought"/>
</d:navigationTree>
```

The docId attribute on the d:import element is of course the id of the navigation document to be imported.

5.4.2.6 Query node

It is possible to dynamically insert nodes by including a query, for example:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:doc id="26" label="Introduction"/>
```

```
  <d:doc id="32" label="Hot Stuff">
    <d:doc id="15" label="Fire"/>
    <d:query q="select name where $somefield='hot'"/>
  </d:doc>
</d:navigationTree>
```

The selected value, in this example "name", will be used as the node label.

Since the query is embedded in an XML file, don't forget that you might need to escape certain characters, e.g. < should be entered as <

Queries embedded in a navigation tree are executed while building the internal model of the navigation tree. This internal model is cached and shared for all users, it is only updated when relevant changes happen in the repository. So if queries contain items that can change on each execution (such as the result of a CurrentDate() call), these will not work as expected.

5.4.2.6.1 Selecting multiple values

If you select multiple values in the query, then group nodes will be created for the additional selected values, for example:

```
select documentType, $Category, name where true
```

With this query, group nodes will be created per value of documentType, within that per value of $Category, and then finally document nodes with the name as label.

5.4.2.6.2 Selecting link values

When selecting a link value (as not-last value), a document node will be created instead of a group node.

5.4.2.6.3 Selecting multi-value and hierchical values

It is allowed to select multi-value and/or hierarchical values. Selecting a hierarchical value will cause the creation of a navigation hierarchy corresponding to the hierarchical path. For multi-value values, nodes will be created for each of the values.

5.4.2.6.4 Re-sorting nodes

Inside the d:query element, you can insert d:column elements specifiying options for each selected value (= each column in the query result set). The number of d:column elements is not required to be equal to the number of selected values (if there are more columns than selected values, the additional columns are ignored). The syntax is:

```
<d:query ...>
  <d:column sortOrder="none|ascending|descending" visibility="..."/>
  ... more d:column elements ...
</d:query>
```

Both the sortOrder and the visibility attributes are optional.

Note that it is also possible to use the "order by" clause of the query to influence the sort order, however with multi-value values or hierarchical values with paths of varying length, it might be needed to have the nodes sorted after tree building. Specifying both an order by clause and sortOrder's on the d:column attributes is not useful, it will only cause extra time to build up the navigation tree.

5.4.2.6.5 Nesting nodes inside a query node

It is possible to nest any sort of node inside a query node, including query nodes themselves. The nested nodes will be executed once for each result in the query, and inserted at the deepest node created by the result set row. If there are any multivalue fields among the selected values in the query, this will be done multiple times.

Attributes of nested nodes can refer to the current result row and its selected values using `${...}` syntax. Available are: `${documentId}`, `${branchId}`, `${languageId}`, and each selected value can be accessed using `${1}`, `${2}`, ...

When query nodes are nested inside query nodes, `${../1}` syntax can be used to refer to values from higher-level query nodes.

If the expressions are inside a "q" attribute of a query node, the value will be automatically formatted appropriately for use in queries. For example, correct date formatting or escaping of string literals. Quotes will be added automatically as necessary, so you can just do something like `$MyStringField = ${3}`.

If the expression is used in the url attribute a link node, the value will be automatically URL-encoded (using UTF-8).

In all other cases, a simple "toString" of the value is inserted.

When an expressions refers to a non existing value (e.g. `${5}` when there are less than 5 selected values), the expression will be left untouched (e.g. the output will contain ${5})

If a selected value is multivalue or hierarchical (or both) it is currently not made available for retrieval in expressions.

To insert `${` literally, use the escape syntax `\${`.

5.4.2.6.6 useSelectValues attribute

If you want to select some values in a query to make them accessible to child nodes, but don't want these values to be used for constructing nodes in the tree, the useSelectValues attribute of the d:query element can be used. The value of this attribute is a number specifying how many of the selected values should be used (counting from the left). This value may be null, in which case the current query node itself will not add any nodes to the navigation tree (this only makes sense if the query node has child nodes).

5.4.2.6.7 Default visibility

The query element can have a visibility attribute to specify the default visibility, for cases where the visibility is not specified for individual columns.

5.4.2.6.8 Filter variants

The query element can have an optional attribute called filterVariants with value true or false. If true, the query results will be automatically limited to the branch and language of the navigation document.

5.4.2.6.9 Example: query-import navigation trees

If you want to dynamically import navigation trees using a query, you can use the following construct:

```
<d:query q="select id where documentType = 'Navigation' order by name"
useSelectValues="0">
  <d:import docId="${documentId} branch="${branchId}" language="${languageId}"/>
</d:query>
```

5.4.2.6.10 Example: generating link nodes

```
<d:query q="select name where true" useSelectValues="0">
  <d:link label="Search on google for ${1}"
          href="http://www.google.com/search?q=${1[1]}"
</d:query>
```

5.4.2.7 Separator node

A separator node is simply a separating line between two nodes in the navigation tree. Its syntax is:

```
<d:separator/>
```

Multiple sibling separator nodes, or separator nodes appearing as first or last node within their parent, are automatically hidden.

5.4.2.8 Associating a navigation tree with collections

If you want to automatically limit the result of queries in the navigation tree to documents contained by one ore more collections, you can add a collections element as first child of the navigationTree element:

```
<d:navigationTree xmlns:d="http://outerx.org/daisy/1.0#navigationspec">
  <d:collections>
    <d:collection name="MyCollection"/>
  </d:collections>
  <d:doc id="26" label="Introduction"/>
  <d:doc id="32" label="Hot Stuff">
    <d:doc id="15" label="Fire"/>
    <d:query q="select name where $somefield='hot' order by name"/>
  </d:doc>
</d:navigationTree>
```

5.4.2.9 Node nesting

The doc, group, query, separator, link and import nodes can be combined and nested as you desire, with the exception that separator and import can't have child elements.

Any other elements besides the ones mentioned here are prohibited, as is text in between the nodes.

5.4.3 Implementation notes

The Navigation Manager is implemented as an extension component running inside the repository server. It has its own HTTP+XML interface and remote Java API.

5.5 Faceted Browser

5.5.1 Introduction

The Daisy Wiki includes a faceted browser which allows for faceted navigation[2] through the repository. The faceted browser shows the distinct values for selected properties (facets) of the documents in the repository, and allows to search for documents by selecting values for these facets. This technique is quite common in many websites, but Daisy's faceted browser makes it very easy to add it to your site.

A somewhat bland demo (i.e. only using system properties) of the faceted browser can be found on the main cocoondev.org site[3].

To use the faceted browser, you need to create a small configuration file in which you list the facets (document properties) to use. You can have multiple faceted navigation configurations.

5.5.2 Howto

Faceted navigations are defined on a per-site level. In the directory for the site, create a subdirectory called "facetednavdefs" if it does not already exist. Thus the location for this directory is:

```
<wikidata directory>/sites/<sitedir>/facetednavdefs
```

In this directory, create a file with the extension ".xml", for example "test.xml". The content of the file should be something like this:

```
<facetedNavigationDefinition xmlns="http://outerx.org/daisy/1.0#facetednavdef">
  <options>
    <limitToSiteCollection>false</limitToSiteCollection>
    <limitToSiteVariant>true</limitToSiteVariant>
    <additionalSelects>
      <expression>variantLastModified</expression>
      <expression>variantLastModifierLogin</expression>
    </additionalSelects>
    <defaultConditions>true</defaultConditions>
    <defaultOrder>documentType ASC, name ASC</defaultOrder>
  </options>
  <facets>
    <facet expression="documentType"/>
    <facet expression="collections"/>
    <facet expression="lastModifierLogin"/>
  </facets>
</facetedNavigationDefinition>
```

About the content of this file:

The options limitToSiteCollection and limitToSiteVariant speak pretty much for themselves, they define whether the query should automatically limit to documents belonging to the collection, branch and language of the current site. If you want to include the collection or branch/language as facets to search on, then you put the respective options to false, otherwise to true. In this

example, since we included *collections* in the list of facets, we put the limitToSiteCollection option to false.

The <facet> elements list the different facets on which the user can browse. The expression attribute contains an identifier as used in the Daisy Query Language (page 63). Thus to include document fields, use $*fieldname*.

<additionalSelects> is an optional element which adds extra identifiers to the select clause of the query which is sent to the repository. Identifiers are the same ones found in the query language and are set as a list of expression elements. The 'name' and 'summary' identifiers are always the first two identifiers found in the query.

The <defaultConditions> element is optional, and contain a set of a set of query conditions to limit the set of documents on which the faceted navigation will be done, for example:

```
<defaultConditions>$someField = 'abc' and $someOtherField='def'</defaultConditions>
```

The <defaultOrder> element is also optional. This element is used to set the default order in which search results will be sorted. The syntax is the same as the order by clause of the query language.

The faceted navigation definition file is validated against an XML Schema, so don't put any additional elements in it or validation will fail.

Once you have saved this file, you can use the faceted browser immediately to browse on the defined facets (a restart of the Daisy Wiki is not needed). The faceted browser is accessed with an URL of this form:

```
http://localhost:8888/daisy/yoursite/facetedBrowser/test
```

In which you need to replace "yoursite" with the name of your site and "test" with the name of the file you just created, without the ".xml" extension.

5.5.3 Usage

5.5.3.1 Faceted browser initialisation

You can define different initialisations for the faceted browser in the faceted navigation definition file. This can be done using an optionsList element.

```
<facetedNavigationDefinition xmlns="http://outerx.org/daisy/1.0#facetednavdef">
  <optionsList defaultOptions="standard">
    <options id="standard">
      <limitToSiteCollection>false</limitToSiteCollection>
      <limitToSiteVariant>true</limitToSiteVariant>
      <defaultConditions>true</defaultConditions>
    </options>
    <options id="doctype">
      <limitToSiteCollection>false</limitToSiteCollection>
      <limitToSiteVariant>true</limitToSiteVariant>
      <defaultConditions>documentType='{request-param:docType|SimpleDocument}'<
/defaultConditions>
      <defaultOrder>documentType ASC</defaultOrder>
 </options>
  </optionsList>
  <facets>
    <facet expression="documentType"/>
    <facet expression="collections"/>
```

```
    <facet expression="lastModifierLogin"/>
  </facets>
</facetedNavigationDefinition>
```

If you wish to be able to choose from a range of different options without having to make different definition files you can use **optionsList** element. It contains a list of different **options** definitions which must be identified using the *id* attribute on the options element. In order to know which set of options should be used by default you must set the *defaultOptions* attribute to the *id* of one of the options elements.

After having defined your optionsList you will probably want to specify one of the options there. This can be done by adding a request parameter in the url. It would look something like this :

```
http://localhost:8888/daisy/yoursite/facetedBrowser/test?options=doctype
```

In the definition you will also find this

```
{request-param:docType|SimpleDocument} -->
{request-param:request-parameter-name|default-request-parameter-value}
```

If the specified request parameter exists the {...} will be substituted by the parameter value. In case that no such parameter exists the default value will be used. In the url our example will look a bit like this :

```
http://localhost:8888/daisy/yoursite/facetedBrowser/test?options=doctype&
docType=SomeDocumentType
```

5.5.3.2 Showing the navigation tree

If you wish to have the navigation tree displayed in the faceted browser you can specify the navigation path as a request parameter (**activeNavPath**). Here is an example :

```
http://localhost:8888/daisy/yoursite/facetedBrowser/test?activeNavPath=/path/to/facetedB
rowser
```

The presence of the parameter will convey your wish to see the navigation tree and set the active navigation node to the specified path.

5.5.3.3 Using an alternative stylesheet

If you wish to use a different stylesheet for the faceted browser than the one found in <skin-dir> /xslt/faceted_browser.xsl, then you can specify this in faceted navigation definition file. Your file might look something like this

```
<facetedNavigationDefinition xmlns="http://outerx.org/daisy/1.0#facetednavdef">
  <stylesheet src="daisyskin:facetednav-styling/myfaceted_browser.xsl"/>
  <options>
   ...
```

Lets follow the example above. First create a directory in your skins directory with the name 'facetednav-styling'. Create a file with the name 'myfaceted_browser.xls' in your freshly created directory. The contents of the file could be something like this

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://outerx.org/daisy/1.0"
  xmlns:p="http://outerx.org/daisy/1.0#publisher"
```

```
  xmlns:n="http://outerx.org/daisy/1.0#navigation"
  xmlns:i18n="http://apache.org/cocoon/i18n/2.1"
  xmlns:daisyutil="xalan://org.outerj.daisy.frontend.util.XslUtil"
  xmlns:urlencoder="xalan://java.net.URLEncoder">
<!-- Import the original stylesheet -->
  <xsl:import href="daisyskin:xslt/faceted_browser.xsl"/>
<!-- The customization -->
  <xsl:template name="content">
    <h1>My own faceted browser</h1>
    <div class="facetbrowser-resultcount"><xsl:value-of
select="d:facetedQueryResult/d:searchResult/d:resultInfo/@size"/> document(s) found.<
/div>
    <br/>
    <xsl:call-template name="options"/>
    <br/>
    <br/>
    <xsl:call-template name="results"/>
    <xsl:call-template name="javascript"/>
  </xsl:template>
</xsl:stylesheet>
```

In the simple example above the title on the faceted browser page was changed and the link to query page was removed. See how the original faceted browser styling was used as a base stylesheet. This stylesheet can be found here :

```
<daisy_home>/daisywiki/webapp/daisy/resources/skins/default/xslt/faceted_browser.xsl
```

Have a look in there to get an idea of what you can customize.

5.5.3.4 Defining discrete facets

If you fear that your facet has too many values to be displayed in an fashionable manner you can define discrete facets. These facets will place values in a series of ranges. There are 3 types of discrete facets :

- **STRING**
  Used for string type values. Words are sampled by the sequence of letters they have in common.
  Properties :

  - This type accepts one property, *threshold*. This is the minimum amount of values for the facet before values start to be grouped. The amount of groups that will then be available are then maximum threshold+1 .

- **DATE**
  Used for date type values. The dates are sampled based on the specified spread.
  Properties :

  - *threshold* (same as above)
  - *spread*. How the ranges are spread out. Ranges are spread out on a logarithmic scale. This property allows specification of the magnitude of the spread. By default this is a magnitude of 1.0 which is a linear spread.

- **NUMBER**
  Used for number types.

Properties :

- threshold (same as above)
- spread (same as above)

When you use the type attribute you tell Daisy that values from this facet can be discrete. Here is an example :

```
<facets>
<!-- A date discrete facet with a parabolic spread. -->
  <facet expression="$SomeDate" type="DATE" threshold="7" spread="2.0">
    <properties>
      <property name="threshold" value="7"/>
      <property name="spread" value="2.0"/>
    </properties>
  </facet>
 <!-- Number discrete facet with a linear spread -->
  <facet expression="$SomeNumber" type="NUMBER">
    <properties>
      <property name="threshold" value="7"/>
    </properties>
  </facet>
  <facet expression="$SomeString" type="STRING">
    <properties>
      <property name="threshold" value="7"/>
    </properties>
  </facet>
</facets>
```

5.5.4 Futher pointers

Faceted Classification Discussion mailing list[4].

5.6 URL space management in the Daisy Wiki

5.6.1 Overview

The URL space of the documents when published through a Daisy Wiki site is related to the hierarchical navigation tree of the site. With *URL space* we mean the URLs that get assigned to documents, or the other way around, what URL you need to enter in the location bar of your web browser to get a certain document. This document goes into some details about how all this works.

URL stands for Uniform Resource Locator. It is how resources (documents etc) on the world wide web are addressed, in other words the widely-known http://host.com/some/path things.

5.6.2 The (non-)relation between the Daisy repository and the URL space

The Daisy repository is a flat (non-hierarchical) document-store. It isn't necessarily web-related, and hence doesn't define, dictate or influence how documents are actually published on the web, including how these documents will map to the URL space.

Recall that documents in the repository are identified by a unique, numeric ID. The uniqueness is within one repository, it is a sequence number starting at 1, not a global unique identifier.

5.6.3 URL mapping

The URL mapping in the Daisy Wiki is based on the hierarchical navigation tree. This means that when a document is requested, the navigation tree is consulted to resolve the path. The other way around, when publishing documents, the logical daisy:<document-id> links that occur in documents that are stored in the repository are translated to the path at which they occur in the navigation tree (if a document occurs at multiple locations, the first occurrence -- in a depth-first traversal -- is used).

5.6.3.1 Relation between the navigation tree and the URL space

Each nested node in the navigation tree becomes a part of the path in an URL. The name of the part of the path is the ID of the navigation tree node. What this ID is depends on the type of node:

- for document nodes, by default it is the document ID, unless a custom node ID is specified in the navigation tree
- for group nodes, it is the id of the node specified in the navigation tree, if the id is not specified, a default one is generated (something like g1, g2, and so on)

If you want to have more readable URLs, it is recommended to assign node IDs in the navigation tree. With *readable URLs* we mean URLs containing meaningful words instead of automatically assigned numbers.

5.6.3.2 Importance of readable URLs?

It is in no way required to assign custom node IDs in the navigation tree. You only need to do this if you want to have readable, meaningful URLs.

Some advantages of having readable URLs is:

- the page may be easier to find (higher ranked) by web search engines such as Google,
- you can guess what the page is about simply by looking at the URL. In contrast, when it is simply a number, that doesn't tell much.

However, URLs containing the raw document IDs also have their advantages:

- you don't have to think about how to call the navigation tree nodes.
- they are more robust to changes in the navigation tree. If you move nodes in the navigation tree (which is usually a very common thing to do), when the URL paths end on the numeric document ID, the document can still be found by using the document ID. This is a result of a general rule when designing URLs: the less meaningful information you put in them, the less likely they are going to break. (When renaming or moving nodes with custom IDs, it is possible to let the old location redirect to the new one. This is currently not possible directly in Daisy, but can be configured in Apache when using Apache in front of Daisy)

It is a good idea to standardise on some conventions when naming navigation tree nodes. For example, use always lower case and separate names consisting of multiple parts with dashes.

If all you want to have are some shortcut URLs for certain documents, independent of where they occur in the navigation tree, you can run Apache in front of the Daisy Wiki and configure redirects over there.

5.6.3.3 How URL paths are resolved in the Daisy Wiki

When a request for a certain path comes in, the Daisy Wiki will ask the navigation tree manager to lookup that path in the navigation tree for the current site. There are a number of possible outcomes:

- the node described by the path exists in the navigation tree and identifies a document. This is the more common case. In this case, the Daisy Wiki will go on to display that page. If a specific branch and language of the document were requested, different from the site's default branch and language or different from the branch and language specified in the navigation tree node, then the site-search algorithm described below is used.
- the node described by the path exists in the navigation tree but identifies a group node. In this case the Daisy Wiki will redirect to the first document child of that node (that is found by doing a depth-first traversal of the group node -- thus first descending child group nodes when encountered).
- the node does not exist in the navigation tree, again multiple possibilities:
    - the path ends on a number: then this number is interpreted as a document ID. The site-search algorithm described below is then used to determine what to do.
    - the path does not end on a number: a ResourceNotFoundException is thrown, resulting to an error page in the browser.

5.6.3.3.1 Site-search algorithm

The site search algorithm is used each time when a document might be more suited for display in the context of another site, thus when the document has not been found in the current site's navigation tree.

The sites that will be considered in the search can be configured in the siteconf.xml (page 155) using the `<siteSwitching>` element, whose syntax is as follows:

```
<siteSwitching mode="stay|all|selected">
  <site>...</site>
  ... more <site> elements ...
</siteSwitching>
```

The mode attribute takes one of these values:

- `stay`: always stay in the current site, no other sites will be searched
- `all`: consider all available sites (in alphabetical order)

- `selected`: consider only a subset of sites, listed using the `<site>` child elements. The content of each `<site>` element should be the name of a site (the name of a site is the name of the directory in which it is defined).
  This mode is recommended when you have a large number of sites (to improve performance), or when you have a number of sites that are related and you don't want the user to be redirected outside this set of sites. It can also be useful when you want to change the order in which sites are considered.

The site-search algorithm works as follows:

- It loops over all considered sites. If the document variant occurs in the navigation tree of the site, the browser will be redirected to this site and navigation tree path.
- If the end of the sites list is reached and no site has been found where the document occurs in the navigation tree, the browser will be redirected to the first site in the list for which the collection, branch and language matched. If there is no such site, the document will be displayed in the current site.

5.6.3.4 Not all documents must appear in the navigation tree

As a consequence of the above described resolving mechanism, any document can be accessed in the repository even if it does not occur in the navigation tree. Simply use an URL like:

```
http://host/daisy/mysite/<document-id>
```

In which <document-id> is the ID of the document you want to retrieve.

After each document URL you can add the extension .html, thus the above could also have been:

```
http://host/daisy/mysite/<document-id>.html
```

By default, the Daisy Wiki generates links with a .html extension, since this makes it easier to download a static copy of the site to the file system (otherwise you could have files and directories with the same name, which isn't possible).

5.7 Document publishing

5.7.1 Document styling

5.7.1.1 Introduction

The Daisy Wiki allows to customize the styling of documents by mean of an XSLT. This custom styling is typically performed depending on the document type.

5.7.1.2 The Input XML

The input of the stylesheets is an XML document which has a structure as shown below. This is not an extensive schema containing every other element and attribute, but those that you'll need most often.

```
<document
    isIncluded="true|false"
    displayContext="standalone|something else"
    xmlns:d="http://outerx.org/daisy/1.0"
    xmlns:p="http://outerx.org/daisy/1.0#publisher">

  <context .../>
  <p:publisherResponse>
    <d:document xmlns:d="http://outerx.org/daisy/1.0"
        id="..."
        name="..."
        [... various other attributes ...] >

      <d:fields>
        <d:field typeId="..." name="..." label="..." valueFormatted="..."
                 [... other attributes and children ...]>
        ... more fields ...
      </d:fields>

      <d:parts>
        <d:part typeId="..." mimeType="..." size="..." label="..."
daisyHtml="true/false">
          [... HTML content of the part including html/body if @daisyHtml=true ...]
        </d:part>
        ... more parts ...
      </d:parts>

      <d:links>
        <d:link title="..." target="..."/>
        ... more links ...
      </d:links>

      [... customFields, lockInfo, collectionIds ...]
    </d:document>
  </p:publisherResponse>
</document>
```

The `isIncluded` attribute on the document element indicates if this document is being published as top-level document or for inclusion inside another document. Sometimes you might want to style the document a bit different when included.

Similarly, the `displayContext` attribute on the document element gives a hint toward the context in which a document is being displayed. The value "standalone" is used for cases where the document is displayed by itself, rather than as part of an aggregation. The value of the displayContext comes from the p:preparedDocument (page 103) instruction in the publisher request.

The context element is the same as in the layout.xsl input (page 191). It provides access to various Wiki-context and user information. Before Daisy 1.5, the context element was not available, only a user element. The user element is still included for compatibility (not shown here) but will eventually be removed.

The p:publisherResponse element then contains the actual document (and possibly related information) to be published. It is the result of the p:preparedDocuments publisher request (page 103). It will always contain the basic d:document element but can contain additional information if a custom publisher request is used.

5.7.1.3 Expected stylesheet output

The output of the XSLT should be an embeddable chunk of HTML (or XSL-FO in the case of PDF). Thus no <html> and <body> elements, but something which can be inserted inside <body > (or inside a , a <td >, etc). Where the produced output will end up depends on the

stylesheet creating the general page layout, or in the case of included documents, the location of the inclusion.

5.7.1.4 Where the stylesheets should be put

The stylesheets should be placed in the following directory:

```
<wikidata directory>/resources/skins/<skin-name>/document-styling/<format>
```

In which `<skin-name>` is the name of the skin you're using (by default: "default"), and `<format>` either `html` or `xslfo`. Thus for the default skin, for HTML, this becomes:

```
<wikidata directory>/resources/skins/default/document-styling/html
```

The stylesheet should be named (case sensitive):

```
<document-type-name>.xsl
```

5.7.1.5 Example 1: styling fields in a custom way

Suppose we have a document type called "TestDocType" with a " SimpleDocumentContent" part , and two fields called "field1" and "field2". The default layout will first place the parts, then the fields (in a table), and then the out-of-line links (if any).

The stylesheet below shows how to put the fields at the top of the document:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://outerx.org/daisy/1.0">

  <xsl:import href="daisyskin:xslt/document-to-html.xsl"/>

  <xsl:template match="d:document">
    <h1 class="daisy-document-name"><xsl:value-of select="@name"/></h1>

    <p>
      Hi there! Here's the value of field1:
      <xsl:value-of select="d:fields/d:field[@name='field1']/@valueFormatted"/>
      and field 2:
      <xsl:value-of select="d:fields/d:field[@name='field2']/@valueFormatted"/>
    </p>

    <xsl:apply-templates select="d:parts/d:part"/>
    <xsl:apply-templates select="d:links"/>
    <!-- xsl:apply-templates select="d:fields"/ -->
  </xsl:template>

</xsl:stylesheet>
```

To minize our efforts, we import the default stylesheet and only redefine what is needed. For comparison, the default template for d:document looks as follows:

```
<xsl:template match="d:document">
  <h1 class="daisy-document-name"><xsl:value-of select="@name"/></h1>
  <xsl:apply-templates select="d:parts/d:part"/>
  <xsl:apply-templates select="d:links"/>
  <xsl:apply-templates select="d:fields"/>
</xsl:template>
```

This new stylesheet should be saved as:

```
<wikidata directory>/resources/skins/default/document-styling/html/TestDocType.xsl
```

Now surf to a document based on TestDocType, and you should see the result.

5.7.1.6 Example 2: styling parts in a custom way

In this example, suppose we have a document type called "Article" with parts "Abstract" and "Body". We would like to render the abstract in a box. The below stylesheet shows how this can be done.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://outerx.org/daisy/1.0">

  <xsl:import href="daisyskin:xslt/document-to-html.xsl"/>

  <xsl:template match="d:document">
    <h1 class="daisy-document-name"><xsl:value-of select="@name"/></h1>

    <div style="margin: 20px; padding: 10px; border: 1px solid black; background-color:
#ffd76c">
      <xsl:apply-templates select="d:parts/d:part[@name='Abstract']"/>
    </div>
    <xsl:apply-templates select="d:parts/d:part[@name='Body']"/>

    <xsl:apply-templates select="d:links"/>
    <xsl:apply-templates select="d:fields"/>
  </xsl:template>

</xsl:stylesheet>
```

5.7.2 Document information aggregation

5.7.2.1 Introduction

When a document is published in the Wiki, the Wiki will retrieve the document using a publisher request (page 90), and style it using a document-type specific stylesheet (page 180) (or fall back to a default stylesheet).

The information available to this stylesheet is basically just the document with its parts and fields (and some wiki-context information such as the user etc.). Sometimes it might be useful to show additional information with the document.

For example, suppose you have a field "Category". When a document is published, you would like to show at the bottom of the document a list of documents which have the same value for the Category field as the document that is being published.

This can be done by making use of custom publisher request for these documents. The basic reference information on this can be found in the publisher documentation (page 90). Here we will have a look at how this applies to the Wiki using a practical example.

If you want to try out the example described here, then make a field Category (string), create a document type having this field (its name does not matter), and create a few documents of this document type, of which at least some share the same value for the Category field.

For none of the changes described here, it is required to restart the repository server or wiki.

5.7.2.2 Creating a publisher request set

The first thing to do is to define a new publisher request set. In the daisydata directory (*not wikidata directory*), you will find a subdirectory called "pubreqs":

```
<daisydata directory>/pubreqs
```

In this directory, create a new subdirectory, for the purpose of this example we will call it "foobar":

```
<daisydata directory>/pubreqs/foobar
```

In this directory, we need to create three files:

- A mapping file which defines when to use which publisher request
- A default publisher request
- A publisher request to be used for the documents having a Category field

Let's start with the mapping file. Create, in the foobar directory, a file named `mapping.xml`, with the following content:

```
<?xml version="1.0"?>
<m:publisherMapping xmlns:m="http://outerx.org/daisy/1.0#publishermapping">
  <m:when test="$Category is not null" use="categorized.xml"/>
  <m:when test="true" use="default.xml"/>
</m:publisherMapping>
```

This mapping tells that when the document has a field Category, the publisher request in the file categorized.xml should be used. In all other cases, the second m:when will match and the publisher request in the file default.xml will be used. The expressions here are the same as used in the query language (and thus as in the ACL).

Instead of checking on the existence of a field, a more common case is to check on the document type. For this you would use an expression like `documentType = 'MyDocType'`.

Create (in the same foobar directory) a file called default.xml with the following content:

```
<?xml version="1.0"?>
<p:publisherRequest xmlns:p="http://outerx.org/daisy/1.0#publisher">
  <p:prepareDocument/>
  <p:aclInfo/>
  <p:subscriptionInfo/>
</p:publisherRequest>
```

This is the same publisher request as is normally used, when you do not bother to create custom publisher requests.

Now we arrive at the most interesting: the publisher request for documents having a Category field. Create a file called categorized.xml with the following content:

```
<?xml version="1.0"?>
<p:publisherRequest xmlns:p="http://outerx.org/daisy/1.0#publisher"
styleHint="categorized.xsl">
  <p:prepareDocument/>
  <p:aclInfo/>
  <p:subscriptionInfo/>

  <p:group id="related">
    <p:performQuery>
      <p:query>select name where $Category = ContextDoc($Category) and id !=
ContextDoc(id)</p:query>
    </p:performQuery>
  </p:group>
```

```
</p:publisherRequest>
```

Note the difference with the publisher request in default.xml. We have now included a query which retrieves the documents with the same value for the Category field, but excluding the current document. The only purpose of the p:group element is to make it possible to distinguish this query from other queries we might add in the future.

Also note the styleHint attribute. This is an optional attribute that can be used to indicate the stylesheet to be used (instead of relying on the document-type specific styling).

5.7.2.3 Telling the Wiki to use the new publisher request set

In the Wiki, the publisher request set to be used can be specified per site (in the siteconf.xml) or can be changed for all sites (in the global siteconf.xml). Either way, this is done by adding a < publisherRequestSet> element in the siteconf.xml. For this example, we will change it globally, thus we edit:

```
<wikidata directory>/sites/siteconf.xml
```

And as child of the root element (the order between the elements does not matter), we add:

```
<publisherRequestSet>foobar</publisherRequestSet>
```

It can take a few seconds before the Wiki notices this change, but you do not need to restart the Wiki for this. If you would now go looking at documents with a Category field, they would still look the same as before, as we have not yet adjusted the stylesheets to display the new information.

5.7.2.4 Creating a stylesheet

The styling is just the same as with regular document-type specific styling, however the styling here is not specific to the document type but rather driven by the publisher request. In the publisher request we used the `styleHint` attribute to tell the Wiki it should use the `categorized.xsl` stylesheet. Other than that, everything is the same as for document-type specific styling. Thus we need to create a file `categorized.xsl` at the following location:

```
<wikidata directory>/resources/skins/default/document-styling/html/categorized.xsl
```

Since these stylesheets are in the same location as the document type specific stylesheets, care should be taken that their names do not conflict with document types (unless on purpose of course). If your custom publisher requests are related to the document type, it is not needed to specify the styleHint attribute as the normal document type specific styling will do its job.

Here is an example categorized.xsl:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://outerx.org/daisy/1.0"
  xmlns:p="http://outerx.org/daisy/1.0#publisher">

  <xsl:import href="daisyskin:xslt/document-to-html.xsl"/>

  <xsl:template match="d:document">
```

```
    <h1 class="daisy-document-name"><xsl:value-of select="@name"/></h1>
    <xsl:apply-templates select="d:parts/d:part"/>
    <xsl:apply-templates select="d:links"/>
    <xsl:apply-templates select="d:fields"/>

    <hr/>
    Other documents in this category:
    <ul>
      <xsl:for-each select="../p:group[@id='related']/d:searchResult/d:rows/d:row">
        <li>
          <a href="{$documentBasePath}{@documentId}?branch={@branchId}&
amp;language={@languageId}">
            <xsl:value-of select="d:value[1]"/>
          </a>
        </li>
      </xsl:for-each>
    </ul>

    <xsl:call-template name="insertFootnotes">
      <xsl:with-param name="root" select="."/>
    </xsl:call-template>
  </xsl:template>

</xsl:stylesheet>
```

And that's it.

5.7.3 Link transformation

This section is about the transformation of links in the Daisy Wiki from "daisy:" to the public URL space.

5.7.3.1 Format of the links

It might be good to review the format of the links first. The structure of a Daisy link is:

```
daisy:docid@branch:language:version#fragmentid
```

The branch, language and version are all optional. Branch and language can be specified either by name or ID. The version is typically a version ID, or the string "live" (default) or "last" (to link to the last version). The fragment identifier is of course also optional.

A fragment identifier is used to point to a specific element in the target document.

If the branch and language are not mentioned, they are defaulted to be the branch and language of the document containing the link (and thus not to the default branch and language of the Daisy Wiki site).

A link can also consist of only a fragment identifier, to link to an element within the current document:

```
#fragmentid
```

The document addressed by a link is called the *target document*.

5.7.3.2 When and what links are transformed

The link transformation happens after the document styling XSLT has been applied.

The transformation applies to links in the following places:

- the `href` attribute of the `<a>` element
- the `src` attribute of the `<img>` element
- the content of `<span class="crossref">` elements

5.7.3.3 Input for the document styling XSLT

Since the link transformation happens after the document styling, the document styling XSLT can influence the linking process (see "Linking directly to parts" below). For this, it is useful to know that the publisher (page 90) leaves some information on links about the target document they link to. An annotated link looks like this:

```
<a href="daisy:123" p:navigationPath="/info/123">
  <p:linkInfo documentName="..." documentType="...">
    <p:linkPartInfo id="..." name="..." fileName="..."/>
  </p:linkInfo>
  see this
</a>
```

The whitespace and indenting is added here for readability, in reality no new whitespace is introduced (since the whitespace inside inline elements is significant).

The p:navigationPath attribute gives the path in the navigation tree where the document occurs. This attribute is only added if such a path exists, and if the navigation tree is known (i.e. if it is specified to the publisher, which in the Daisy Wiki is always the case). You should usually leave the p:navigationPath attribute alone, the link transformation process will use it to make the link directly point to the 'good' navigation location (afterwards, it will remove the p:navigationPath attribute).

The p:linkInfo element is only added if the target document exists and is accessible (i.e. the user has read permissions on it). It specifies the name of the target document and the name of its document type. Also, for each part in the document, a p:linkPartInfo element is added. Its id and name attribute specify the part type ID and part type name of the part. The fileName attribute is only added if the part has a file name.

It is the responsibility of the document styling XSLT to remove the p:linkInfo element. This is very simple with an empty template that matches on this element (as is the case in the default document-to-html.xsl).

5.7.3.4 Linking directly to parts

By default, the links will be transformed to links that point to the target document. This seems obvious, but sometimes it is desirable to link directly to the data of a part of the target document, for example for images. The link transformation process can be instructed to do so by leaving special attributes on the link element (`<a>` or `<img>`) in its namespace:

```
http://outerx.org/daisy/1.0#linktransformer
```

which is typically associated with the prefix `lt`.

The attributes are:

- `lt:partLink`: specifies the name of the part to link to, e.g. ImageData

- `lt:fileName`: optionally specifies a filename to append at the end of the URL path (otherwise the file name is always 'data')

Examples of how to put this to use can be found in the document-to-html.xsl, more specifically look at how images and attachments are handled there.

5.7.3.5 Branch and language handling

If the branch and language differ from those specified in the site configuration, branch and language request parameters will be added.

5.7.3.6 Fragment ID handling

Fragment identifiers are prefixed with a string identifying the target document. This is needed because it is possible to publish multiple Daisy documents in one HTML page (e.g. using document includes), and the same element ID might be used in multiple documents, giving conflicts.

The format of the prefix is `dsy<docid>_`, an example prefixed fragment identifier looks like this:

```
#dsy123_hello
```

in which "123" is the ID of the target document, and "hello" the original fragment identifier target.

To make this work, the actual element IDs in Daisy documents are also prefixed with `dsy<docid>_`.

5.7.3.7 Disabling the link transformer

If you want the link transformer to leave a certain link alone, add an attribute `lt:ignore="true"` on the link element.

5.7.4 Document publishing internals

Here we will eventually add a complete description of how the process of getting a document published in the Daisy Wiki works behind the curtains.

For now, please see the Cocoon GT presentation (page 0) on this subject.

5.8 Daisy Wiki Skinning

Customising the look and feel of the Daisy Wiki is possible through:

- configuration of an existing skin, via the skinconf.xml
- creation of a custom skin

Daisy ships with one skin called `default`.

The skin to use is configurable on the level of a site in the siteconf.xml (page 155) file. Thus different sites can use different skins.

Pages not belonging to a particular site (such as the login screen, the sites index page, etc.) use a globally configured skin, defined in the global siteconf.xml file:

```
<wikidata directory>/sites/siteconf.xml
```

If this file does not exist, the skin called `default` will be used. The content of the global siteconf.xml file should be like this:

```
<siteconf xmlns="http://outerx.org/daisy/1.0#siteconf">
  <skin>default</skin>
</siteconf>
```

5.8.1 skinconf.xml

5.8.1.1 Introduction

The skinconf.xml file is simply an XML file which is merged in the general XML stream and is available to the XSLT stylesheets, most specifically the layout.xsl (page 191). This allows to pass configuration information to XSLT stylesheets. The actual supported configuration will be dependent on the XSLT, and thus on the skin. The supported configuration for the default skin is given below.

A skinconf.xml file can be put in the site (page 155) directory, or in the global sites directory:

```
<wikidata directory>/sites/skinconf.xml
```

If a site doesn't have its own skinconf.xml file, it will fall back to using the global one.

5.8.1.2 default skin skinconf.xml

```
<skinconf>
  <logo>resources/local/mylogo.png</logo>
  <daisy-home-link>Daisy Home</daisy-home-link>
  <site-home-link>Site Home</site-home-link>
</skinconf>
```

Each of the parameters (= XML elements) is optional.

The parameters quite speak for themselves:

- `logo`: the path of the logo to put in the left-top corner. If you don't create a custom skin, you can for example put it at the following location:

  ```
  <wikidata directory>/resources/local
  ```

- `daisy-home-link`: alternative text for the "Daisy Home" link (in the top-right corner)
- `site-home-link`: alternative text for the "Site Home" link (in the top-right corner)

It can take up to 10 seconds before changes made to a skinconf.xml file are noticed.

5.8.2 Creating a skin

5.8.2.1 The anatomy of a skin

A skin consists of a set of files: CSS file(s), images, XSLT stylesheets, and possibly others which are grouped below one directory. The directory containing the skins is located at:

```
<wikidata directory>/resources/skins
```

The name of the skin is the name of the directory below the skins directory. On a blank Daisy install, this skins directory will be empty. The default skin can be found in:

```
<DAISY_HOME>/daisywiki/webapp/daisy/resources/skins/default
```

5.8.2.2 Creation of a dummy skin

Daisy has a **fallback mechanism** between skins, which means that a new skin can be created based on an existing skin. This makes the initial effort of creating a skin very small.

As an example, suppose you want to create a skin called `coolskin`. The minimal steps to do this are:

1. Create a directory for the skin:

   ```
   <wikidata directory>/resources/skins/coolskin
   ```

2. In this newly created directory, put a file called `baseskin.txt` containing just one line like this:

   ```
   default
   ```

   (this should be the very first line in that file)

   This specifies that the new skin will be based on the default skin. This means that any file which is not available in the `coolskin` skin, will instead be taken from the default skin. This allows a skin to contain only copies of those files that it wants to change.

Although there is no directory called `default` in the skins directory, the system will transparently fall back to the skins directory in the Daisy Wiki webapp (mentioned above).

3. Modify one or more siteconf.xml (page 155) files to use the new skin. For the non-site specific pages (login screen, index page, ...) this is:

   ```
   <wikidata directory>/sites/siteconf.xml
   ```

   Or for a specific site, the siteconf.xml file in the directory of that site.

Basically, you now have created a new skin, although it doesn't do anything yet. If you hit refresh in your browser, you will still see the same.

5.8.2.3 Customising the new skin

Now you can start customising the skin by copying files from the default skin and adjusting them. The two most important files, which allow to change most of the global look of the Daisy Wiki, are these:

1. the `<skindir>/css/layout.css` file
2. the `<skindir>/xslt/layout.xsl` file

If you only want to do smaller changes like changing some colours and fonts, you should get around by only copying the docstyle.css file to your new skin and adjusting it. (note: to change the logo, you can use the skinconf.xml (page 189) mechanism)

The layout.xsl file builds the global layout of a page, thus how everything 'around' the main content shoud look. The input format of the XML that goes in the layout.xsl can be found here (page 191).

5.8.3 layout.xsl input XML specification

This is the layout.xsl input contract.

```
<page>
  <!-- The context element is usually produced by the PageContext class
       (but the layout.xsl doesn't care about this of course) -->
  <context>
    <!-- Information about the Daisy Wiki version -->
    <versionInfo version="..." buildHostName="..." buildDateTime="..."/>

    <!-- The mountPoint is everything of the URI path that comes before
         the part matched by the Daisy sitemap. By default, this is /daisy -->
    <mountPoint>...</mountPoint>

    <!-- The current 'version mode' -->
    <versionMode>live|last</versionMode>

    <!-- The site element specifies some information about the current Daisy Wiki site,
         this is of course only required when working in the context of a site. -->
    <site
      name="..."
      title="..."
      description="..."
      navigationDocId="..."
      collectionId="..."
      collection="..."
      branchId="..."
      branch="..."
      languageId="..."
      language="..."/>

    <!-- skinconf: the contents of the skinconf.xml file of the current site,
         or of the global skinconf.xml file in case the current page does is
         outside the context of a site. -->
    <skinconf/>

    <!-- user: information about the current user -->
    <user>
      <name>...</name>
      <login>...</login>
      <id>...</id>
      <activeRoles>
        <role id="..." name="..."/>
        (... multiple role elements ...)
      </activeRoles>
```

```
      <updateableByUser>true|false</updateableByUser>
      <availableRoles default="name of default role">
        <role id="..." name="..."/>
      </availableRoles>
    </user>

    <!-- layoutType: the type of layout that the layout.xsl must render.
         Tree possibilities:
            - default: the normal layout, possible with navigation tree,
                       page navigation links, links to other variants, etc.
            - mini: minimalistic layout, which shouldn't put the page content
                    inside a table (this layout is used by the editor screen,
                    and the HTMLArea in IE doesn't work when put inside a table.
            - plain: a layout that doesn't display anything beside the content.
    -->
    <layoutType>default|mini|plain</layoutType>

    <!-- request: some info about the request:
              - uri: the full request URI, including query string
              - method: GET, ...
              - server: scheme + host + port number if not 80
                        to which the HTTP request has been sent
    -->
    <request uri="..." method="..." server="..."/>

    <!-- skin: name of the current skin. Can be useful to use in paths to
         resources (images, css, js, ...) -->
    <skin>...</skin>
  </context>

  <!-- pageTitle: a title for the page, this is what comes inside the html/head/title
       element and thus in the title bar of the users' browser. This element may
       contain mixed content (e.g. i18n tags) so its content must be copied entirely,
       not just the string value. -->
  <pageTitle>...</pageTitle>

  <!-- layout hints (optional element):
        wideLayout: if there is no navigation tree and you want to make use of
                    the maximum available width, specify true.
        needsDojo: if you want dojo to be loaded on a non-cforms page, add this
                   attribute.
  -->
  <layoutHints wideLayout="true" needsDojo="true"/>

  <!-- A hierarchical navigation tree as produced by the navigation manager. Optional.
-->
  <n:navigationTree/>

  <!-- ACL evluation information of the navigation tree. Optional, if present this is
       used to display the navigation view/edit links -->
  <navigationInfo>
    <d:aclResult ... />
  </navigationInfo>

  <!-- pageNavigation: these are page-specific links (actions). As for the pageTitle,
       the link/title element may contain mixed content.
       pageNavigation is optional. -->
  <pageNavigation>
    <link [needsPost="true"]>
      <title>...</title>
      <path>...</path>
    </link>
    (... more link elements ...)
  </pageNavigation>

  <!-- availableVariants: a list of other variants of the document displayed
       on the current page. Optional. -->
  <availableVariants>
    <variants>
      <variant href="..."
               branchName="..."
               languageName="..."
```

```
                [current="true"]/>
    </variants>
  </availableVariants>

  <!-- content: the actual content to be displayed on the page. The content
       of this element must be copied over literally into the resulting output. -->
  <content>
   ...
  </content>

  <!-- extraMainContent: additional content, which must be placed below the normal
       content, but only in the default layoutType. This is currently used to have
       comments displayed in the default layout but not in the plain layout. -->
  <extraMainContent>
    ...
  </extraMainContent>

  <!-- extraHeadContent: additional content which will be copied inside
       the <head> element -->
  <extraHeadContent>
    ...
  </extraHeadContent>

</page>
```

5.8.4 The daisyskin and wikidata sources

5.8.4.1 Introduction

The *daisyskin* and *wikidata* sources are Cocoon (Excalibur/Avalon) sources. Sources in Cocoon are additional schemes you can use in URLs. For example, for the daisyskin source this means you can use URLs of the form "daisyskin:something".

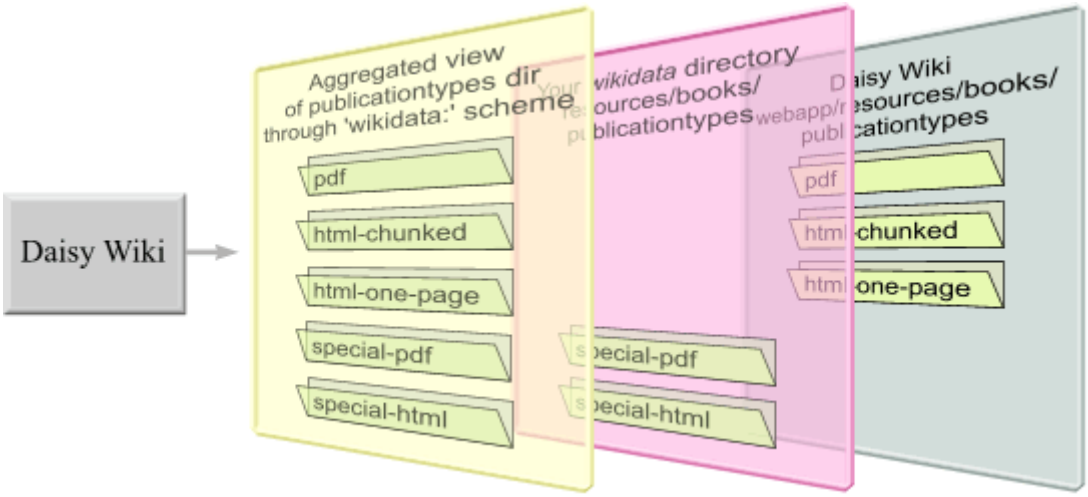
What makes the daisyskin and the wikidata sources special is that they have *fallback behaviour*. If a file is not found in a first location, it is searched for in another location. It is as if directories are transparently layered on top of each other.

The wikidata source only performs fallback between the wikidata directory and the Daisy Wiki webapp, while the daisyskin sources additionally can fall back to 'base' (parent) skins.

5.8.4.2 wikidata source

The wikidata source performs fallback between a wikidata directory and the Daisy Wiki webapp. Thus if a file is not found in the wikidata directory, it is taken from the Daisy Wiki webapp.

For example, the publication types for the books are accessed via the wikidata scheme. The Daisy Wiki has some built-in default publication types, and in addition you can define your own ones in the wikidata directory. By using the wikidata scheme, the Daisy Wiki sees both without requiring any special effort.



In some cases, it can be useful to refer directly to the file in the webapp. This can be done using a "(webapp)" hint in the URL:

```
wikidata:/(webapp)/path/to/file
```

This would basically be the same as not using the wikidata source at all. The advantage is that the 'wikidata' source knows about the location of the webapp and we keep working through the same abstraction, which at some point might prove useful.

One case where this might be useful is when you have an XSL in the wikidata dir that hides the original XSL in the webapp dir. In that case, you could include the hidden XSL in the hiding XSL like this:

```
<xsl:include href="wikidata:/(webapp)/path/to/my.xsl"/>
```

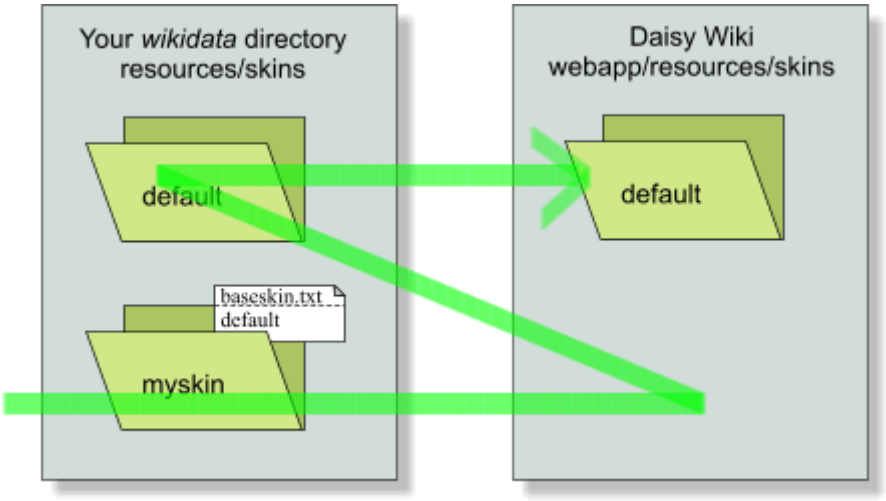
How the location of the actual wikidata directory used by the wikidata scheme is determined, is described in Specifying the wikidata directory location (page 317).

5.8.4.3 daisyskin source

The daisyskin source:

- takes the specified file automatically from the current skin (the 'current skin' changes depending on the current site)
- has fallback behaviour:
    - if a file doesn't exist in a skin, it will be searched for in its base skin (recursively, untill a skin without base skin is encountered).
    - if a file doesn't exist in the wikidata directory, it will be searched for in the Daisy Wiki webapp.

The fallback first goes from wikidata to webapp, and then to the next base skin, as illustrated by the arrow in the image below.



5.8.4.3.1 URL structure

Most of the time one will use relative URLs like for example:

```
daisyskin:images/foo.png
```

This will load the file images/foo.png from the directory of the current skin. If this file isn't available in the skin, it will be loaded from this skins' base, or the base skin of that skin, and so on, until it is found (or not found).

Additionally, the URL can specify the name of the skin using the following syntax:

```
daisyskin:/(myskin)images/foo.png
```

This can be useful in you have an XSL in a custom skin, in which you want to import the corresponding XSL from the default skin.

And as last option, it is also possible to indicate that the file should always be taken from the webapp, without checking the wikidata directory:

```
daisyskin:/(myskin)(webapp)images/foo.png
```

5.8.4.3.2 Specifying the base skin

The base skin of a skin is specified by having a file called baseskin.txt in the directory of the skin. This file should simply contain one line (with no blanks before it) with the name of the base skin. The presence of the baseskin.txt file is optional, a skin isn't required to have a base skin. Recursive base skin references are of course not allowed (and are detected).

5.8.4.3.3 Note about caching

The content of the baseskin.txt files is cached, this cache is updated asynchronously, with a certain frequency, by default every 10 seconds. So if you change the content of a baseskin.txt file, it can take up to 10 seconds before the change will be noticed. The file-changed-checks are performed by a component called the ActiveMonitor, see the cocoon.xconf to change its execution interval.

Other than this, nothing is cached so file additions or removals are detected immediately.

5.9 Query Styling

5.9.1 Overview

By default, query results are rendered as a table. It is however possible to customize the styling of the query results. This is done by supplying a `style_hint` option in the query, for example:

```
select name where true option style_hint = 'bullets'
```

The style hint 'bullets' is included as a sample with Daisy, it styles the results as a bulleted list, taking the first selected value (here 'name') as the text to put next to the bullet.

5.9.2 Implementing query styles

Style hints are implemented using XSL. Implementing a style hint is a matter of adding a template to the `query-styling-(html|xslfo).xsl` files of a skin.

If you want the query styles to be available to all skins, it is recommended to add them to the query-styling-*.xsl of the default skin. Otherwise, add them to the skin of your choice.

The stylesheets for the query styling are located in the following directory:

```
<wikidata directory>/resources/skins/<skin-name>/query-styling
```

Thus for the default skin, this is:

```
<wikidata directory>/resources/skins/default/query-styling
```

If this directory would not yet exist in your wikidata directory, you can simply create it.

Then in that directory, create a file called query-styling-html.xsl (if it doesn't exist already). The following shows an example of what the XSL could look like.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://outerx.org/daisy/1.0">

  <xsl:include href="daisyskin:/(default)(webapp)query-styling/query-styling-html.xsl"/>

  <xsl:template match="d:searchResult[@styleHint='enum']">
    <ol>
      <xsl:for-each select="d:rows/d:row">
        <li>
          <a href="{$searchResultBasePath}{@documentId}.html?branch={@branchId}&
amp;language={@languageId}">
            <xsl:value-of select="d:value[1]"/>
          </a>
        </li>
      </xsl:for-each>
    </ol>
  </xsl:template>

  <!-- Add other query-styling templates here -->

</xsl:stylesheet>
```

Here we implemented the query styling for the style hint "enum". It is basically the same as the bullets styling, but using an <ol> instead of <ul>.

Also note the special xsl:include. By creating this query-styling-html.xsl file for the default skin, we hide the one in the default skin in the webapp directory. However, we can import it in the

current XSL using this instruction. The "(default)" in the href indicates the name of the skin (if not specified, the 'current skin' is used), and the "(webapp)" tells to explicitely use the file from the webapp directory, not the one from the wikidata directory (which would be the file we have created here, so if we didn't add the (webapp) we would have a recursive include).

Usually when an XSLT or one of the XSLTs included/imported by it changes, Cocoon should reload it. However, it seems this only works for the first level of includes, if included files themselve again include other XSLTs, it seems to stop working. Therefore changes to the file `query-styling-html.xsl` are not immediately reflected in the wiki. There is an easy work-around to avoid having to restart the wiki In order to let the changes take effect: simply touch an XSLT which Cocoon will check for changes, in this case the searchresult.xsl (which is a first-level include in document-to-html.xsl). In case you applied document type specific styling, you have to touch the stylesheet you created for that purpose (e.g. `<wikidata directory>/resources/skins/default/document-styling/html/mydoctype.xsl`). ('Touching' means updating the last modification time of the file, e.g. by saving it in an editor, or on unix using the `touch` command).

If you would add a query-styling-html.xsl to another skin than the default skin, and would like to include the stylings from the default skin, you can use the following include instruction in the XSL:

```
<xsl:include href="daisyskin:/(default)query-styling/query-styling-html.xsl"/>
```

Note that the href doesn't contain the "(webapp)" part, so that the daisyskin source will first check for a query-styling-html.xsl for the default skin in the wikidata directory.

For more information on the daisyskin source, see its documentation (page 193).

5.10 Daisy Wiki PDF Notes

5.10.1 Introduction

Daisy generates PDFs by feeding XSL-FO[5] (a vocabulary for specifying formatting semantics) to Apache FOP[6] (and XSL-FO processor). The PDFs are generated on the fly when a user requests them.

5.10.2 Images

FOP caches images, normally indefinitely. Daisy clears FOPs' image cache from time to time, by default every 5 minutes (this is configurable in the cocoon.xconf file). So if you upload a new version of an image, it can take up to 5 minutes before appearing in the PDF. Closing and reopening your browser would also give immediately the new variant, since images are retrieved on a per-session basis (the sessionid is encoded in the image URL).

5.10.3 Links

In PDF output, link URLs are mentioned in footnotes.

5.10.4 Layout limitations

Below we list the known limitations in PDF output.

5.10.4.1 Table column widths

FOP can't auto-balance column widths like web browsers do. Therefore, by default all columns in a table are made the same width, which might give undesirable results. However, you can assign custom column widths in the editor through the table settings dialog. Next to the absolute widths such as cm and mm, there's also the special unit * (star). This is for proportional widths. For example, assigning the columns of a two-column table the widths 1* and 2* respectively, will make the first column take one third of the width and the second two thirds.

Another possibilities is to assign a class to the table and use that to perform appropriate styling in a custom skin or document type specific XSLT.

5.10.4.2 Text flow around images

Text flow around images is not supported.

5.10.4.3 Table cell vertical alignment

Table cell vertical alignment is not supported.

5.11 Daisy Wiki Extensions

5.11.1 Introduction

The Daisy Wiki has some hooks to add your own functionality. You build extensions by building on top of Cocoon and making use of the available Daisy repository API (plus extension components such as the navigation manager and the publisher). To develop extensions, you don't need a Java development environment or knowledge, though you can if you want. The Daisy Wiki contains some samples to get you started.

Some examples of what you could do using extensions:

- define pipelines that produce HTML blurbs and include them using the `cocoon:` protocol in your documents. This can be used to retrieve data from RDBMS databases, RSS feeds, or any system with an accessible interface.
- define entirely new pages:
    - which can be styled using the same skin as the rest of the Daisy Wiki (or not, as you prefer)
    - which can contain custom forms (see guestbook example)
    - which can display arbitrary combinations of documents and navigation trees

Daisy Wiki extensions are, just as the name implies, just extensions to the Daisy Wiki. If you want to develop a completely custom site, it is better to start from "scratch", albeit reusing some basic groundwork for interacting with the repository server and getting pages published. We are still working on providing a solution for this though, so in the meantime you can get along using the Daisy Wiki with skinning can extensions.

5.11.2 Basics

5.11.2.1 Where to put extensions

Extensions can be site-specific or shared by all sites.

Site-specific extensions are placed in a directory called `cocoon` in the directory of the site, thus:

```
<wikidata directory>/sites/<sitename>/cocoon
```

Cross-site extensions (shared by all sites) are placed in a directory called `cocoon` in the `sites` directory, thus:

```
<wikidata directory>/sites/cocoon
```

5.11.2.2 How extensions work

The Cocoon[7] framework, on which the Daisy Wiki is build, has a concept called sitemaps. A **sitemap** describes how to handle a request, i.e. when a request is handled by Cocoon, Cocoon will consult the sitemap in order to know what it has to do. One sitemap can delegate to another sitemap, and that is how Daisy Wiki extensions are integrated in the main Daisy Wiki.

More specifically, all URLs within a site starting with **ext/** are delegated to the extension sitemap: first the site-specific one and then the cross-site one.

Thus for the public URLs, this means all paths below:

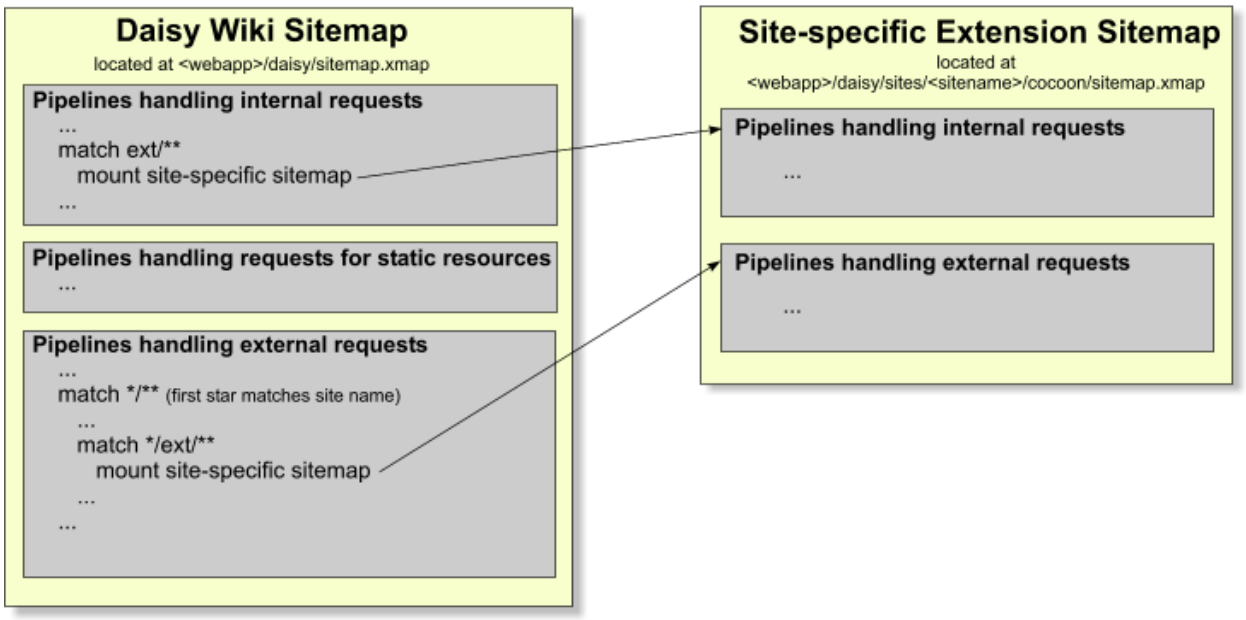
```
http://localhost:8888/daisy/<sitename>/ext/
```

are handled by the extension sitemaps.

5.11.2.2.1 Some more details about the sitemap mounting

(If you are not familiar with Cocoon, don't be bothered that you don't understand all of this yet.)

The Daisy Wiki sitemap consists of two main blocks: one that handles internal requests and one that handles external requests. Each of these two blocks contains a matcher that will forward requests to the extension sitemap. This is illustrated in the following diagram.



In the diagram, the site-specific sitemap is also split into a part handling external requests and a part handling internal requests, but that is free choice of the implementer of the site-specific sitemap. An advantage of putting the internal requests pipeline above the one for external requests, is that for external requests the internal part will be skipped immediately, leading to a faster matching process. Another reason is that in the Daisy Wiki main sitemap, the pipeline for handling external requests starts with various initialisation actions, such as for determining the locale, looking up the site, ... By putting a separate pipeline for internal requests above the one for external requests, duplicate execution of that code is prevented in case of internal requests.

So in general, it is advisable that pipelines which will only be called internally (e.g. included in a document via the cocoon: protocol, or pipelines called from flow-logic), are put in the internal pipelines section.

5.11.3 Getting started

5.11.3.1 Creating your first extension

Here we show how to create a very simple and mostly useless "hello world" extension, just to illustrate some basics. We will create a Cocoon pipeline which generates a blurb of HTML showing "Hello world" and then show how to include that in a Daisy document.

5.11.3.1.1 Create a directory for your extension

We will create a cross-site extension. To make it easy to add multiple extensions in a modular way, we'll organize the extensions in subdirectories. Therefore, edit or create a sitemap.xmap file at the following location:

```
<wikidata directory>/sites/cocoon/sitemap.xmap
```

For a *site-specific* extension, follow the same instructions but use <wikidata directory> /sites/<sitename>/cocoon as the base location in which you do everything.

and put the following in this sitemap.xmap (if there is already something in it, just replace it with this):

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
  </map:components>

  <map:views>
  </map:views>

  <map:resources>
  </map:resources>

  <map:pipelines>

   <map:pipeline>
     <map:match pattern="*/**">
       <map:act type="ResourceExists" src="{1}/sitemap.xmap">
         <map:mount check-reload="yes" src="{../1}/sitemap.xmap" uri-prefix="{../1}"/>
       </map:act>
     </map:match>
   </map:pipeline>

 </map:pipelines>

</map:sitemap>
```

Then create a new subdirectory for your extension, lets say we call it `mytest`:

```
<wikidata directory>/sites/cocoon/mytest
```

With the sitemap.xmap file we just created, if you now request an extension URL starting with "mytest/", the cocoon/sitemap.xmap will try to forward the request processing to a sitemap located at cocoon/mytest/sitemap.xmap (this is what the map:mount does). We will create this sitemap in the next section.

5.11.3.1.2 Create a sitemap

In the above created directory (mytest), create a file called `sitemap.xmap` with the following content:

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
  </map:components>

  <map:views>
  </map:views>

  <map:resources>
  </map:resources>

  <map:pipelines>

   <map:pipeline type="noncaching">
     <map:parameter name="outputBufferSize" value="8192"/>

     <map:match pattern="hello">
       <map:generate src="hello.xml"/>
       <map:transform src="hello.xsl"/>
       <map:serialize type="xml"/>
     </map:match>
```

```
    </map:pipeline>

  </map:pipelines>
</map:sitemap>
```

The sitemap.xmap file is used by Cocoon to decide how to handle a request. This sitemap specifies that if the path matches "hello" (at least, the part of the path stripped from the part leading to this extension), then an XML-producing pipeline is executed which starts by reading and parsing the file hello.xml, transforming the parsed XML using the hello.xsl XSLT, and then finally serializing the result back to an XML document in the form of a byte stream (which is sent to the browser, or whoever made the HTTP request)

5.11.3.1.3 Create the file hello.xml

Still in the same directory, create a file called `hello.xml` with the following content:

```
<?xml version="1.0"?>
<hello>
  <helloText>Hello world!</helloText>
</hello>
```

5.11.3.1.4 Create the file hello.xsl

Also in the same directory, create a file called `hello.xsl` with the following content:

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:template match="hello">
    <div style="border: 1px solid blue;">
      <xsl:value-of select="helloText"/>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

For those unfamiliar with XSLT, this stylesheet defines a template which is executed when a `hello` element is encountered in the input. In that case, a `div` element is outputted with as content the value of the `helloText` element that is a child of the current `hello` element.

5.11.3.1.5 Try it out in your browser

Surf to the following URL (or equivalent), substituting `<sitename>` with the name of your Daisy Wiki site:

```
http://localhost:8888/daisy/<sitename>/ext/mytest/hello
```

If your browser shows XML (such as Firefox or IE), you will see this:

```
<div style="border: 1px solid blue;">Hello world!</div>
```

5.11.3.1.6 Include this in a Daisy document

To include this piece of HTML in a Daisy document:

- create or edit a document

- in the block-style dropdown, choose *Include*
- the paragraph switches to an include-style paragraph
- enter `cocoon:/ext/mytest/hello` for the content of the paragraph

Background detail: "`cocoon:`" URLs allow to make internal Cocoon requests, thus these requests don't go over HTTP, nor is the result of the called pipeline serialized in between (Cocoon pipelines are SAX-based, thus are not based on byte streams but rather XML-representing events).

- save the document, you should see the text "Hello world!" in a blue box in your document

5.11.3.2 Further pointers

You can now explore the various samples for further inspiration.

You'll see that many samples make use of Cocoon flowscript, which is basically Javascript with some Cocoon-specific APIs included (which also includes an advanced flow-control feature called continuations, but see the Cocoon documentation for that).

Daisy provides some additional functions for usage in flowscript, see the daisy-util.js reference (page 203).

In the sitemap, a line like this calls a function defined in a .js file:

```
<map:call function="minimalRss"/>
```

The function can then do some work (such as gathering data) and finally calls the sitemap again to show a page, using the `sendPage` function.

Many examples also make use of the **Publisher**, which is an extension component running inside the repository server. Its purpose and request format is described on its own page (page 90).

5.11.4 daisy-util.js API reference

To make the Daisy Wiki context and functionality easily available from flowscript (javascript), a small integration library called daisy-util.js is available.

5.11.4.1 Importing

To use it, add the following on top of the javascript file:

```
cocoon.load("resource://org/outerj/daisy/frontend/util/daisy-util.js");
```

5.11.4.2 The Daisy object

To avoid naming conflicts, all provided functions are wrapped in a "Daisy" object. To use any of the functions, get a reference to the Daisy object like this:

```
var daisy = getDaisy();
```

and then call the functions on the daisy object. The getDaisy() method returns a singleton instance of the Daisy object, so don't be afraid of calling it as often as you like (rather than passing the daisy instance around).

5.11.4.3 Functions

5.11.4.3.1 daisy.getRepository()

Returns the Repository object for the current user. Using the repository object, you can perform any repository operation, from creating documents to performing queries. See the javadoc API included in the binary Daisy distribution.

This is the method you should most often use, the methods below are for special cases.

5.11.4.3.2 daisy.getRepository(login, password)

Returns the Repository object for an arbitrary user.

5.11.4.3.3 daisy.getGuestRepository()

Returns the Repository object for the guest user. This can be useful if you want to do something specifically as guest.

If the current user is "not logged in" (which means, is automatically logged in as guest user), you can simply use the getRepository() method.

5.11.4.3.4 daisy.getPageContext(repository)

The repository argument is optional, by default the repository object of the current user will be used.

The PageContext object is an object encapsulating various context information and is streamable as XML. An instance of PageContext is usually passed on to the view layer and streamed by a JX template (see the `${pageContext}` in the JX templates). This automatically generates the context element that is required as input to the layout.xsl (page 191).

5.11.4.3.5 daisy.resolve(uri)

Resolves an URL to a more absolute form. There is nothing Daisy-specific about this method, but this can be useful when reusing a pipeline from the main Daisy Wiki sitemap while specifying a file located in the current extension. This is used in the guestbook sample.

5.11.4.3.6 daisy.getHTMLCleaner(configFilePath)

Returns a HTMLCleaner object. The configFilePath is optional, and specifies the path to a htmlcleaner.xml file. By default, the default htmlcleaner.xml of the Daisy Wiki will be used.

The HTMLCleaner object allows to clean up and normalize HTML in the same way as is otherwise done when editing through the default document editing screen. The most common usage is:

```
var daisy = getDaisy();
var content = "<html><body>Hello world!</body></html>";
var htmlcleaner = daisy.getHTMLCleaner();
var data = htmlcleaner.cleanToByteArray(content);
```

The data object, which is a Java byte array, can then be supplied to the setPart() method of the document object (see the repository API javadoc).

5.11.4.3.7 daisy.performPublisherRequest(pipe, params, publishType, repository)

Performs a publisher request (page 90). The publisher request is build by executing the Cocoon pipeline specified by the pipe argument. The params argument is supplied as "viewData" to this pipeline. Such a pipeline will typically use the JX template generator.

The publisher request built from the pipeline is then executed by the publisher.

If the request contained any `<p:preparedDocuments>` elements having an attribute `applyDocumentTypeStyling="true"`, then the prepared documents included in this element will be extracted, document type specific styling will be applied to them, and the result will be put aside in a request attribute for later merging by use of the DaisyIncludePreparedDocuments transformer.

The result of executing the publisher is returned as a SAXBuffer object, this is an object which can easily be streamed in a pipeline using the JX template generator (or a file generator with src="xmodule:flow-attr:something").

The publishType argument identifies the kind of document type specific styling that should be applied, this is either `html` or `xslfo`. This argument is only relevant if the publisher request contains any `<p:preparedDocuments applyDocumentTypeStyling="true"/>` requests.

The repository argument is optional, by default the repository object of the current user is used.

5.11.4.3.8 daisy.buildPublisherRequest(pipe, params)

This builds a publisher request, the same way as is done by daisy.performPublisherRequest, but doesn't execute it. This can be useful for debugging purposes: call buildPublisherRequest with the same parameters as you would call performPublisherRequest, and dump it to, for example, the console:

```
var publisherRequest = daisy.buildPublisherRequest(pipe, params);
java.lang.System.out.println(publisherRequest);
```

Of course, if you do this for debugging purposes, be sure to remove these calls afterwards!

BTW, for the curious, to execute the publisher request yourself, the scenario is like this:

```
var daisy = getDaisy();
var publisherRequest = daisy.buildPublisherRequest(....);
var repository = daisy.getRepository();
var publisher = repository.getExtension("Publisher");
publisher.processRequest(publisherRequest, ..some sax contenthandler..);
```

If you use the `performPublisherRequest` method, the "..some sax contenthandler.." will be a ContentHandler that automatically does the document type specific styling stuff.

5.11.4.3.9 daisy.getSiteConf()

Returns an object of the following type:

```
org.outerj.daisy.frontend.components.siteconf.SiteConf
```

for the current site. This allows access to various site parameters such as its default collection.

5.11.4.3.10 daisy.getMountPoint()

Returns the part of the URL path leading to the main Daisy sitemap (thus, up until where the URLs are interpreted by the Daisy Wiki). On a default deployment, this is `/daisy`.

5.11.4.3.11 daisy.getDaisyContextPath()

Returns the URL of directory containing the main Daisy sitemap, thus something like `file:/..../daisywiki/webapp/daisy/`. This can be useful to access Daisy Wiki resources from extensions.

5.11.4.3.12 daisy.getDaisyCocoonPath()

Returns the part of the URL path that is interpreted by Cocoon and leads to the Daisy Wiki sitemap. On a default deployment, this is `/daisy`. This is useful to call pipelines in the main Daisy Wiki sitemap.

To illustrate the difference with getMountPoint: if the servlet context path would for example be `/cocoon`, then:

daisy.getMountPoint() would return `/cocoon/daisy`

and

daisy.getDaisyCocoonPath() would return `/daisy`

5.11.4.3.13 daisy.getLocale()

Returns the java.util.Locale object for the current user's locale.

5.11.4.3.14 daisy.getLocaleAsString()

Returns the locale as string.

5.11.4.3.15 daisy.getVersionMode()

Returns the active "version mode", which is either 'live' or 'last'. This indicates which version of documents the user wants to see by default.

The returned object is a WikiVersionMode object, converting it to string gives either 'live' or 'last'.

5.11.5 Document editor initialisation

5.11.5.1 Introduction

Usually users create new documents by choosing a document type on the "New document" page. Sometimes it can be useful to create documents from other places, or to have the editor initialised with certain content (e.g. a field with a value already assigned or so). Here we have a look at the various ways to launch the document editor for creation of a new document.

5.11.5.2 The basics

To open the document editor for the creation of a new document, do a HTTP POST operation to the following URL:

```
http://localhost:8888/daisy/<sitename>/new/edit
```

Of course, change the host name and port number, the mount point ("/daisy") and the sitename to match your situation. This URL needs to be supplied with different parameters depending on what you want to, as described in the next sections.

5.11.5.2.1 startWithGet parameter

In some cases, you may want to redirect to the document editor from some server-side code. In such cases it is not possible to do a HTTP POST operation. Therefore, a special `startWithGet` parameter is available, to be used as follows:

```
http://localhost:8888/daisy/<sitename>/new/edit?startWithGet=true
```

5.11.5.2.2 returnTo parameter

Normally, when the user is done editing, the user will be shown the document that was just edited (or, when pressing cancel and it was a new document, the site's home page will be shown). It is possible to control where the user will be brought to after editing a document by means of a `returnTo` request parameter:

```
http://localhost:8888/daisy/<sitename>/new/edit?returnTo=/some/path/of/your/choice
```

This is useful when the edited document is part of some aggregated display, and you want to bring the user back to the aggregated page, rather then edited document.

5.11.5.3 Create a new document of a certain type

This is the most common case. The editor will be opened with a blank document of a certain type. The document type needs to be specified in a parameter called `documentType`, either by name or ID.

Thus for example, doing a POST to this URL will open a new editor for a SimpleDocument-type document

```
http://localhost:8888/daisy/<sitename>/new/edit?documentType=SimpleDocument
```

Optionally, branch and language parameters can be added if they would differ from the site's default.

5.11.5.4 Create a new document starting from a template document

This allows to create a new document by starting from the content of an existing document in the repository. This is the same as the "duplicate document" functionality that is available in the Daisy Wiki.

The ID of the template document needs to be specified in a parameter called template. Branch and language can optionally be specified, they default to those of the current site.

```
http://localhost:8888/daisy/<sitename>/new/edit?template=123&branch=main&
language=default
```

5.11.5.5 Creating a new document variant

The cases described until now were about creating entirely new documents. Here we look at how to open the editor to add a new variant to an existing document.

The following request parameters need to be specified, all required:

- variantOf: the document ID
- startBranch and startLanguage: specify the existing variant from where to start
- newBranch and newLanguage: specify the new variant to create

The branches and languages can be specified by name or ID.

So an example URL could be:

```
http://localhost:8888/daisy/<sitename>/new/edit?variantOf=123&startBranch=main&
startLanguage=en&newBranch=main&newLanguage=fr
```

5.11.5.6 Creating a new document with custom initialisation

This allows to open the editor with certain data already in the new document. It works as follows:

- create a Daisy Document object, set its properties as desired
- put the created Document object in a request attribute
- and then do an internal redirect to the editor, specifying a parameter `templateDocument` with the name of the request attribute containing the Document object.

This requires that you create a Daisy Wiki extension.

Here is an example:

```
cocoon.load("resource://org/outerj/daisy/frontend/util/daisy-util.js");

function makeNewDocument() {
    var daisy = getDaisy();
    var repo = daisy.getRepository();

    // Create the document
    // The parameters are the document name and the document type name
    var newDoc = repo.createDocument("A new document", "SimpleDocument");

    // Set some initial content in a part
    // The part content must be given as a byte array
    var initialContent = new java.lang.String("<html><body><p>Type something here."
                            + "</p></body></html>").getBytes("UTF-8");
    newDoc.setPart("SimpleDocumentContent", "text/xml", initialContent);

    // Add the document to the site's collection
    var siteCollection = repo.getCollectionManager().getCollection(
            daisy.getSiteConf().getCollectionId(), false);
    newDoc.addToCollection(siteCollection);

    // Store the document in a request attribute. The request attribute can have any
name
    cocoon.request.setAttribute("myDoc", newDoc);
```

```
    // Switch to the editor
    var url = "cocoon:/" + daisy.getDaisyCocoonPath() + "/" +
daisy.getSiteConf().getName()
              + "/new/edit?templateDocument=myDoc&startWithGet=true";
    cocoon.redirectTo(url);
}
```

For those not familiar with extensions yet, lets make the example complete.

Create a subdirectory in `<wikidata dir>/sites/cocoon`, for example called 'newtest'. Then save the above script in a file test.js in the newtest directory. Also in the newtest directory, create a file sitemap.xmap with the following content:

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
  </map:components>

  <map:views>
  </map:views>

  <map:resources>
  </map:resources>

  <map:flow language="javascript">
    <map:script src="test.js"/>
  </map:flow>

  <map:pipelines>

   <map:pipeline>
     <map:match pattern="newtest">
       <map:call function="makeNewDocument"/>
     </map:match>
   </map:pipeline>

 </map:pipelines>

</map:sitemap>
```

You can now call this example by surfing to the following URL:

```
http://localhost:8888/daisy/<sitename>/ext/newtest/newtest
```

(replace <sitename> and other stuff as appropriate. The first 'newtest' corresponds to the name of the 'newtest' directory, the second 'newtest' is the pattern matched in the sitemap.xmap)

5.11.6 Samples

5.11.6.1 Daisy Wiki extension sample: publish document

This sample shows how to publish just one document.

To use, just surf to an URL like:

```
http://localhost:8888/daisy/<sitename>/ext/publishdoc/<documentId>
```

in which you need to replace `<sitename>` with the name of your site and `<documentId>` with the ID of a document.

This simple example can be useful if you want to integrate a "published document" into an external system. Compared to retrieving a document from the default Daisy URLs, this example will not do any redirecting based on the navigation tree, and doesn't translate links in the document based on the navigation tree, which is the behaviour you'll usually prefer for this kind of applications. It's also easy to modify the XSL to create an XML envelope around the document containing any additional information you may need.

Building on this example, you could also do things like composing a page consisting of multiple published documents, adding one or more navigation trees, etc. All this just by extending the publisher request and doing some appropriate XSLing on the result.

5.11.6.2 Daisy Wiki extension sample: RSS include

This sample shows you how to include a live import of an Atom/RSS feed into a document.

Quite often, such feeds include escaped HTML (i.e. with < and > encoded as `<` and `>`), so we need some way to reparse that HTML into proper XML which can be passed through the Daisy publishing pipelines. Luckily, a Cocoon component exists for this exact purpose: the HTMLTransformer which is embedded in the Cocoon HTML Block. The HTML Block isn't included by default in the Daisy distribution, but the compiled jar can be found at the iBiblio Maven repository[8]. Copy cocoon-html-2.1.7.jar[9] into `$DAISY_HOME/daisywiki/webapp/WEB-INF/lib/` and you should be set.

Then, you need to define an extension pipeline which transforms Atom markup into HTML for inclusion into a Daisy document.

Create a directory `myrss` inside `<wikidata directory>/sites/cocoon` and add this `sitemap.xmap` to it:

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
    <map:transformers>
      <map:transformer name="htmltransformer"
                       src="org.apache.cocoon.transformation.HTMLTransformer"/>
    </map:transformers>
  </map:components>
  <map:views>
  </map:views>
  <map:resources>
  </map:resources>

  <map:pipelines>

    <map:pipeline internal-only="true" type="noncaching">
      <map:match pattern="myrss">
        <map:generate src="/some/path/leading/upto/myatomfeed.xml"/>
        <!-- Thanks to Cocoon, this path could also be a URL.
             Keep in mind this URL will be accessed every time
             the including document is retrieved! -->
        <map:transform type="htmltransformer">
          <map:parameter name="tags" value="content"/>
        </map:transform>
        <map:transform type="xalan" src="atom2html.xsl"/>
        <map:serialize type="xml"/>
      </map:match>
    </map:pipeline>

  </map:pipelines>

</map:sitemap>
```

Next, add this XSL stylesheet (`atom2html.xsl`) to the `myrss` directory:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:atom="http://purl.org/atom/ns#">

  <xsl:template match="/">
    <div>
      <xsl:for-each select="/atom:feed/atom:entry">
        <h2><xsl:value-of select="atom:title"/></h2>
        <xsl:copy-of select="atom:content/html/body/*"/>
      <p style="text-align: right; color: #999; font-size: 80%;">
        Originally blogged by <xsl:value-of select="atom:author/atom:name"/> on
        <a href="{atom:link/@href}"><xsl:value-of select="atom:issued"/></a>.
      </p>
      </xsl:for-each>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

The last work is to include the snippet of generated HTML into a Daisy document. Add an "include"-style paragraph to the document content with this content:

```
cocoon:/ext/myrss/myrss
```

That's it! When you save the document, you'll see that the Atom feed is dynamically pulled into the webpage as a partial HTML document.

5.11.6.3 Daisy Wiki extension sample: guestbook

This sample illustrates using a form to collect data (based on Cocoon's form framework, CForms) and creating a document in the repository using the collected data. The theme of the sample is a "guestbook", though we don't want to publicise this as the correct way of building a guestbook.

This sample is not included in the by default deployed cross-site extensions, but must be manually 'installed' (copied) into a certain site, and requires some small customisation. See the directory `<webapp>/daisy/ext-samples` and the README.txt file over there.

5.11.6.4 Daisy Wiki extension sample: navigation aggregation

This example creates a page composed of all documents occurring in a navigation tree, in the order in which they occur in the navigation tree. It is also possible to limit the result to a certain subsection of the navigation tree.

To use, call an URL like:

```
http://localhost:8888/daisy/<sitename>/ext/navaggregator/navaggregator
```

in which you replace `<sitename>` by the name of your site.

To limit to a certain section of the navigation tree, add an `activePath` request parameter:

```
http://localhost:8888/daisy/<sitename>
/ext/navaggregator/navaggregator?activePath=/abc/def
```

You can see the active path in the browser location bar when you move over nodes in a navigation tree, it is the part after the sitename.

The basic example is very simple, it could be extended do to things like:

- adding a table of contents with links to the documents
- supporting getting other navigation trees then the site default one
- support xslfo rendering.

All this isn't that particular difficult to realize either.

5.11.6.5 RSS

RSS feeds are provided in the Daisy Wiki as a cross-site extension (page 198), since there are many different feeds that make sense, and this easily allows to build your own if you don't like the ones provided by default.

The URLs for the default feeds provided by this extension are (replace `<sitename>` with the name of a site):

```
http://localhost:8888/daisy/<sitename>/ext/rss/minimal-rss.xml
http://localhost:8888/daisy/<sitename>/ext/rss/normal-rss.xml
http://localhost:8888/daisy/<sitename>/ext/rss/editors-rss.xml
```

The differences between these feeds are:

- the **minimal** feed includes only the name of the documents with a recently published live version
- the **normal** feed includes only the whole rendered content of the documents with a recently published live version
- the **editors** feed includes both the rendered content and a diff with the previous version (if any) of the documents with a recently created version (regardless of whether the version is published)

The following notes apply to all the feeds.

- The feeds include the documents modified in the last 14 days, with a maximum of 20 items.
- The feeds are cached for 20 minutes, so changes can take up to 20 minutes to show up in the feed (see below for non-cached feeds)
- By default, the feeds are generated for the "guest" user. To specify another user, append `?login=...&password=...` to the URL.
- By default, the feeds are generated for the locale "en-US". To specify another locale, append `?locale=...` to the URL, e.g. `?locale=fr` or `?locale=nl`.
- For the minimal and normal feeds, documents of type Navigation, Image and Attachment are excluded.

Non-cached feeds can be accessed with the following URLs:

```
http://localhost:8888/daisy/<sitename>/ext/rss/minimal-rss-direct.xml
http://localhost:8888/daisy/<sitename>/ext/rss/normal-rss-direct.xml
http://localhost:8888/daisy/<sitename>/ext/rss/editors-rss-direct.xml
```

To have RSS-links appear on the recent changes page, the RSS feeds need to be configured in the skinconf.xml. The default skinconf.xml contains the config for the above RSS feeds, so you can have a look there to see how its done.

5.12 RSS

RSS feeds are provided in the Daisy Wiki as a cross-site extension (page 198), since there are many different feeds that make sense, and this easily allows to build your own if you don't like the ones provided by default.

The URLs for the default feeds provided by this extension are (replace `<sitename>` with the name of a site):

```
http://localhost:8888/daisy/<sitename>/ext/rss/minimal-rss.xml
http://localhost:8888/daisy/<sitename>/ext/rss/normal-rss.xml
http://localhost:8888/daisy/<sitename>/ext/rss/editors-rss.xml
```

The differences between these feeds are:

- the **minimal** feed includes only the name of the documents with a recently published live version
- the **normal** feed includes only the whole rendered content of the documents with a recently published live version
- the **editors** feed includes both the rendered content and a diff with the previous version (if any) of the documents with a recently created version (regardless of whether the version is published)

The following notes apply to all the feeds.

- The feeds include the documents modified in the last 14 days, with a maximum of 20 items.
- The feeds are cached for 20 minutes, so changes can take up to 20 minutes to show up in the feed (see below for non-cached feeds)
- By default, the feeds are generated for the "guest" user. To specify another user, append `?login=...&password=...` to the URL.
- By default, the feeds are generated for the locale "en-US". To specify another locale, append `?locale=...` to the URL, e.g. `?locale=fr` or `?locale=nl`.
- For the minimal and normal feeds, documents of type Navigation, Image and Attachment are excluded.

Non-cached feeds can be accessed with the following URLs:

```
http://localhost:8888/daisy/<sitename>/ext/rss/minimal-rss-direct.xml
http://localhost:8888/daisy/<sitename>/ext/rss/normal-rss-direct.xml
http://localhost:8888/daisy/<sitename>/ext/rss/editors-rss-direct.xml
```

To have RSS-links appear on the recent changes page, the RSS feeds need to be configured in the skinconf.xml. The default skinconf.xml contains the config for the above RSS feeds, so you can have a look there to see how its done.

5.13 Part Editors

5.13.1 Introduction

Starting with Daisy 1.4, the document editor has been changed to make it easy to insert custom editors for parts.

5.13.2 Configuration

The rules for detecting which part editor to use are as follows:

- if there is part specific editor configured, use that
- if the "Daisy HTML" flag of the part type is true, use the HTML editor
- in all other cases, show the upload-editor

To use a custom editor for a certain part, a file called `<partTypeName>.xml` should be created in the following directory:

```
<wikidata directory>/resources/parteditors/
```

The file should contain some XML like the following:

```
<?xml version="1.0"?>
<partEditor xmlns="http://outerx.org/daisy/1.0#parteditor"
        class="...">
  <properties>
    <entry key="...">...</entry>
  </properties>
</partEditor>
```

The `class` property of the partEditor element should contain the class name of a class implementing the following interface:

```
org.outerj.daisy.frontend.editor.PartEditorFactory
```

The properties element is optional and can be used to configure the part editor.

Besides the default editors, there is one example editor included in Daisy: a plain text editor. Suppose you have a part called MyPlainText and you want to use this editor for it, you need to create a file called MyPlainText.xml in the above mentioned directory, and put the following XML in it:

```
<?xml version="1.0"?>
<partEditor xmlns="http://outerx.org/daisy/1.0#parteditor"
        class="org.outerj.daisy.frontend.editor.PlainTextPartEditor$Factory">
</partEditor>
```

Once you save this file, the editor will become immediately active (no need to restart the Wiki). If you do not see the editor, check the cocoon error log.

5.13.3 Implementation info

A part editor is implemented by means of a CForm (Cocoon Form). This consists of a form definition and a form template. To fit into Daisy's document editor framework, there are two things you need to pay attention to:

- The form definition should declare an action with id "dummy", as follows:

```
<fd:action id="dummy">
  <fd:label>dummy action</fd:label>
</fd:action>
```

  This action is needed for the working of the document editor, you simply need to add it and not care about it otherwise (i.e. you do not need to insert it into your form template).
- The form template should not contain a `ft:form-template` element, as it will be embedded inside the document editors' template.

Besides the form definition and template, you need to implement the interface `org.outerj.daisy.frontend.PartEditor` (and corresponding `PartEditorFactory`). This interface defines methods that will be called for constructing, loading and saving of the form.

To learn more, it is best to explore the sources of the default Daisy part editors.

5.14 Internationalisation

5.14.1 Introduction

We welcome translations to new languages, or improvements to the currently available languages. Send translations to the mailing list, or attach them to an issue in Jira[10]. Make sure you use UTF-8 encoding in your files (except for the .properties files -- see below). Also, check on the mailing list if anybody else is working on the same language as you.

Daisy and the Daisy Wiki can be translated to different languages. For this purpose, text to be displayed to users is retrieved from a language-specific file, usually called a *resource bundle*.

5.14.2 Different types of resource bundles

There are three different types of resource bundles used by Daisy:

- for the Cocoon part, these are XML files.
- for messages produced by Java code (the repository server), these are " .property" files
- for Javascript, the resource takes the form of a snippet of Javascript code (.js files)

When editing resource bundles, take care of the encoding of the files, as explained below.

5.14.2.1 Encoding of the XML and .js files

We prefer to keep all the XML and .js (javascript) files UTF-8 encoded, for all languages. Therefore, take care to use the correct encoding when opening and saving files in your text editor.

5.14.2.2 Encoding of the .properties files

The ".properties" files should always be ISO-8859-1 encoded. Characters not supported by ISO-8859-1 can be inserted using Java unicode escapes. A Java unicode escape takes the form "\uHHHH" in which the HHHH are hexadecimal digits specifying the unicode character. Usually you don't want to edit this by hand, the unix recode utility can be handy here:

1. (if it is an existing file) do:

```
recode JAVA..UTF-8 somefile.properties
```

2. then open the file in an editor using UTF-8 as encoding
3. then convert back:

```
recode UTF-8..JAVA somefile.properties
```

5.14.3 Overview of the resource bundle files

Below is an overview of the translatable resources that currently exists. The given directory paths refer to locations inside the source tree, in the binary distribution these locations are a bit different (and the .property files are bundled inside the jar files). See further on this page for practical instructions.

- the Daisy Wiki
  - General

```
applications/daisywiki/frontend/src/cocoon/webapp/
  daisy/resources/i18n/messages_<lang>.xml
```

  - Skin-specific messages

```
applications/daisywiki/frontend/src/cocoon/webapp/
  daisy/resources/skins/default/i18n/messages_<lang>.xml
```

  - Daisy HTMLArea plugins

```
applications/daisywiki/frontend/src/cocoon/webapp/
  daisy/resources/cocoon/forms/htmlarea/plugins/*/lang/<lang>.js
```

  - JS messages

```
applications/daisywiki/frontend/src/cocoon/webapp/
  daisy/resources/js/lang/daisy_edit_<lang>.js
```

- Daisy Wiki initialisation program

```
install/src/java/org/outerj/daisy/install/daisywiki_schema.xml
(this is the same file for all languages)
```

- Java messages

```
applications/daisywiki/frontend/src/java/org/outerj/daisy/
  frontend/components/userregistrar/messages_<lang>.properties
```

- Navigation & book tree editor

```
applications/daisywiki/frontend/src/cocoon/webapp/daisy/
  resources/navtree_editor/lang/navtree_<lang>.js
```

and

```
applications/daisywiki/frontend/src/cocoon/webapp/daisy/
  resources/tree_editor/lang/tree_<lang>.js
```

and

```
applications/daisywiki/frontend/src/cocoon/webapp/daisy/
  resources/book_editor/lang/booktree_<lang>.js
```

- Books functionality

```
applications/daisywiki/frontend/src/cocooon/webapp/daisy/
  books/publicationtypes/common/i18n/messages_<lang>.xml
```

and

```
applications/daisywiki/frontend/src/cocooon/webapp/daisy/
  books/publicationtypes/html-chunked/i18n/messages_<lang>.xml
```

- Dojo widgets

```
applications/daisywiki/frontend/src/cocoon/webapp/daisy/
  resources/js/widget/nls/...
```

- repository server messages:

  - exceptions

```
repository/api/src/java/org/outerj/daisy/repository/messages_<lang>.properties
```

  - identifier titles

```
repository/server/src/java/org/outerj/daisy/query/model/messages_<lang>
.properties
```

  - diff component

```
services/diff/src/java/org/outerj/daisy/diff/messages_<lang>.properties
```

  - workflow

```
services/workflow/server-impl/src/java/org/outerj/daisy/
  workflow/serverimpl/messages_<lang>.properties
```

- email notifications
  - ```
    services/emailnotifier/server-impl/src/java/org/outerj/daisy/
      emailnotifier/serverimpl/formatters/messages_<lang>.properties
    ```
- workflow processes
  - ```
    services/workflow/server-impl/src/processes/*/i18n/*
    services/workflow/server-impl/src/processes/common-i18n/*
    ```
- resources not part of Daisy
  - messages of CForms (mostly validation errors, part of Cocoon)
  - HTMLArea and its "TableOperations" plugin. HTMLArea itself is already translated in many languages, but its TableOperations plugin not.

To make the new language appear on the locale selection page, its ISO language code should be added to the following file:

```
applications/daisywiki/frontend/src/cocoon/webapp/daisy/resources/conf/locales.txt
```

5.14.4 Windows installer

All installation messages of the windows installer are kept inside on single file, which also acts as template for any new languages to be added:

```
distro/windows-installer/installer/lang/english.nsh
```

The NSIS installer does not support Unicode yet, it assumes no encoding and just copies the bytes as is. Therefore the encoding used is mentioned in the comment section in the header of the .nsh language file. In order to display the messages correctly, the default language for non-Unicode applications in the language control panel of your Windows installation has to be set appropriately.

5.14.5 Making or improving a translation

Install the subversion[11] client if you don't have it already.

Check out the Daisy source tree:

```
svn co http://svn.cocoondev.org/repos/daisy/trunk/daisy
```

After editing the files, create a patch by executing the following command in the root of the Daisy source tree:

```
svn diff > mypatch.txt
```

and send the mypatch.txt file to the mailing list.

5.15 User self-registration

People can register themselves as users in the Daisy Wiki. By default, they get assigned the "guest" role. For the default ACL (access control list) that ships with Daisy, this means a registered user will still not be able to edit documents. However, the user will be able to subscribe to notification emails and to add comments to documents.

If you want self-registered users to have a different role(s) assigned after registration, you can configure this in the following file:

```
<wikidata directory>/daisy.xconf
```

In that file, search for the string *UserRegistrar*. On the next few lines you'll see this:

```
    <!-- The names of the roles to which the user should be assigned -->
    <roles>
      <role>guest</role>
    </roles>
    <!-- The default role for the user, should be one of those mentioned in the roles
list. -->
    <defaultRole>guest</defaultRole>
```

which you can adjust according to your wishes.

To have this change take effect, you need to restart the Daisy Wiki.

5.16 Live and staging view

By default, the Daisy Wiki shows the live version of each document, if you want to see other versions you need to explicitly go look at them.

The live/staging switch allows to make that the last version of each document is displayed by default. Or from another point of view, it shows how the Wiki would look like if the last versions of all documents were live.

The live/staging applies to:

- the documents themselves
- images embedded in documents
- documents included inside documents
- queries embedded in documents are executed with the 'search_last_version' option
- navigation trees:
    - the version of the navigation tree document itself is the last one
    - imported navigation tree documents also use the last one
    - if the document name is used as label of a node, the document name from the last version is used
    - queries in navigation trees are executed with the 'search_last_version' option

- in 'last' mode, only document-nodes for which one has read access are visible (versus 'read live' access otherwise)

Since the fulltext index is only maintained for the live content, the fulltext search feature doesn't change when switching to staging mode.

The live/staging mode is stored in the user's session, in other words it applies to the whole Daisy Wiki (all sites), and is not remembered after logging out or closing the browser.

More technically, the live/staging switch is implemented by setting the versionMode attribute in the publisher (page 90) request.

In extensions, you can find out the current mode using the daisy.getVersionMode() method of the Daisy flowscript API (page 203).

5.17 Variables in documents

5.17.1 Introduction

Daisy allows to insert variables in documents, which will be replaced with their actual value during publishing of the documents.

A typical use-case of this is to avoid hard-coding product names in documentation, since the product might be marketed with different names or could change over time.

Another way to look at variables is that they allow for re-use of inline content, while document includes allow for re-use of block content (inline and block are HTML/CSS terminology: inline = things which are stacked horizontally on lines, block = things which are stacked vertically on a page). Be aware though of the limitations, explained further on.

Here are some basic facts about the variables:

- Variable name-to-value mappings are defined in Daisy documents of type Variables, containing a VariablesData part.
- It is possible to configure variable substitution on a per-site level. Thus the same document, displayed in different sites, can have its variables resolved to different values.
- Book publishing also supports variables.
- The actual variable resolving is a feature of the Publisher (page 90), configured using p:variablesConfig (page 109).
- Variables can be inserted in document content (= Daisy-HTML parts) or in the document name. Some effort is done to substitute variables in these document names also when the document name occurs in navigation trees or query results.

5.17.2 Quick variables how-to

These are the basic steps to put variables in action.

Create a new document of type Variables and save it. As you will see, the editor for the variables is simply XML-source based. Take note of the ID of the created document, you'll see it in the URL after saving, something like 55-DSY.

Create one of these files, depending on whether you want to configure variable resolution per-site or for the Wiki as whole:

```
site-specific:
<wiki data dir>/sites/<sitename>/conf/variables.xml

global:
<wiki data dir>/conf/variables.xml
```

And put the following in it:

```
<?xml version="1.0"?>
<variables>

  <variableSources>
    <variableDocument id="55-DSY"/>
  </variableSources>

  <variablesInAttributes allAttributes=”true”/>

</variables>
```

In this XML, **change the value of the id attribute** to the ID of the created variables document.

Allow up to ten seconds for the Wiki to notice this configuration change.

You are now ready to start using variables. Create a new document (e.g. of type Simple Document), on the toolbar of the rich text editor there's a button to insert variables. If you insert one, and save the document, you should see the variable resolved to its value.

5.17.3 Defining variables

5.17.3.1 Creating new variable documents

Variables are defined by creating documents of type Variables, containing the part VariablesData.

Actually the only real requirement is the presence of the VariablesData part, so one could create other types of documents containing this part and use them for variable substitution.

5.17.3.2 Structure of the variable documents

The variables are specified in an XML format, which follows this structure:

```
<v:variables xmlns:v="http://outerx.org/daisy/1.0#variables">
  <v:variable name="var1">value 1</v:variable>
  <v:variable name="var2">value<strong>2</strong></v:variable>
</v:variables>
```

As you can see in the example, variable values can contain mixed content.

The editor will validate the well-formedness and structure of the XML upon saving. When the content of the part is empty (= typcially when creating a new document), some sample XML is inserted to get you started.

5.17.3.3 Editing existing variable documents

If you want to edit the variables in use in the current site, you can use Tools -> Variables to get an overview of the active variable documents.

5.17.3.4 If a variable document is invalid

If during document publishing, a variable document cannot be loaded (e.g. due to invalid XML), no exception will be thrown, rather the variables defined in that document will not be available and an error will be logged to the repository log file.

5.17.3.5 Staging / live mode

Depending on whether the site is in staging or live mode, either the last or live versions of the variable documents will be used.

5.17.3.6 Permissions on the variables documents

The variable documents are filtered based on the read or read live rights (depending on staging / live mode) of the current user on the document.

5.17.4 Configuring Wiki variable resolution

It is possible to define the variable configuration globally for the Wiki, or per site. If the config for a site is missing, it will fall back to the global configuration, if present.

The global variable configuration should be placed here:

```
<wiki data dir>/conf/variables.xml
```

The site-specific variable configuration should be placed here:

```
<wiki data dir>/sites/<sitename>/conf/variables.xml
```

The content of the files is in both cases the same, and follows this structure:

```
<?xml version="1.0"?>
<variables>

  <variableSources>
    <variableDocument id="64-DSY"/>
    <variableDocument id="128-DSY"/>
    <variableDocument id="256-DSY"/>
  </variableSources>

  <variablesInAttributes allAttributes=”false”>
    <element name="img" attributes="daisy-caption,[comma separated list]"/>
    <element name="table" attributes="daisy-caption"/>
  </variablesInAttributes>

</variables>
```

Multiple variable documents can be specified, these will be searched in the given order when resolving variables.

For resolving variables in attributes, one can either explicitely define the attributes for which this should happen, or simply say to do it in all attributes. Explicitely listing the attributes can be

used when performance is a concern (however, the amount of attributes in a Daisy document is typically not very high).

5.17.4.1 When the configuration doesn't seem to be used

The Daisy Wiki only looks for configuration changes every ten seconds. Thus if you count to ten and don't see the result of your configuration changes, there's something wrong.

Problems with errors in the configuration file (e.g. invalid XML) will be logged in the Daisy Wiki log file. They will not cause display of documents to fail.

Problems with loading the variables documents (e.g. don't exist, invalid XML, etc.) will be logged in the repository server log file.

5.17.5 Configuring variable resolution for books

This is done using some book metadata properties (page 238).

5.17.6 Inserting variables in documents

5.17.6.1 Inserting variables in document text.

This can be done by using the “Insert/remove variable” button on the toolbar.

In HTML source, a variable is inserted as `<span class='variable'>variable-name</span>`

5.17.6.2 Inserting variables in attributes and in the document name

This can be done by using the syntax `${variable-name}`. To insert the text `${...` literally, use `$${...`.

Note that the actual attributes which support variable substitution are dependent upon configuration.

5.17.7 Unresolved variables

If a variable cannot be resolved to a value, the following construct is inserted: `<span class='daisy-missing-variable'>[unresolved variable: variable name]</span>`

5.17.8 Limitations of the variables

By putting variables in documents, the documents become simple templates which generate different content depending on the context in which they are displayed. If the user sees a word in a document which results from a variable, and the user searches for that word using the fulltext search, then the document will not be part of the search results since that word doesn't really occur in the document.

Similarly, searches on document names search on the not-variable-resolved document names.

The variable names are part of the fulltext index, so you can search where variables are used.

Variable substitution in document names is currently not performed everywhere in the Wiki. The basic document display (documents, navigation tree, queries embedded in documents) are processed, but in other locations such as the document browser used in the editor etc. the

document names are are unprocessed, since authors are probably more interested to see the raw document name.

5.18 Daisy Wiki Implementation

The Daisy Wiki is build on top of the Apache Cocoon[12] framework. Cocoon is historically strong in XML-based web publishing, which is very useful when implementing a CMS frontend. Next to that, Cocoon also gained powerful webapp development support, in the form of innovative f low-control solutions and a powerful and easy-to-use form handling framework.

Since Cocoon is one of the lesser-known frameworks, an introduction is provided below that should give a head-start into the wonderful world of Cocoon.

5.18.1 So what is Cocoon?

5.18.1.1 Sitemaps and pipelines

Two important concepts of Cocoon are the **sitemap** and **pipelines**. The sitemap is defined in an XML file (called sitemap.xmap) and it is based on the sitemap that Cocoon decides how to handle an incoming request (a browser HTTP request or a programmatic internal request). We'll come back to that in a moment.

A pipeline is an XML-processing pipeline, and consists of three types of components: a generator, followed by one or more transformers and ended by a serializer. These three are connected by SAX events. SAX events represent things you find in an XML document: for example a start tag with attributes, character data, and end tags. A generator produces SAX events, a transformer changes SAX events (one example is an XSLT-transformer), and a serializer serializes the SAX events to a byte stream (an XML document, a HTML document, or possibly an image file if the events represent an SVG picture, or a PDF is the events describe a XSL-FO document).

The sitemap contains matchers that match on request properties, most often this is the URL path. By walking through the sitemap, a pipeline is constructed. As a simple (but very common) example, an extract from a sitemap can look like:

```
<map:match pattern="/pages/mypage">
  <map:generate src="somefile.xml"/>
  <map:transform src="some_stylesheet.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

Suppose you enter in the location bar of your browser a request for " /pages/mypage". When this request is handled by Cocoon, Cocoon will run over the sitemap till it encounters a matcher that matches the path "/pages/mypage". Inside the matcher, it will find the definition of a pipeline to be executed. When it encounters the "map:serialize", it knows all it needs to know, it stops the evaluation of the sitemap and will start executing the pipeline. To compare with Java, you could imagine the sitemap to be one big method where the matchers are if's, and where map:generate, map:transform and map:serialize instructions set/add properties to a pipeline object. After the map:serialize, it is as if there were a return instruction that stops the execution of the method.

The generator used in the example above is the default one (because, unlike the serializer, it is missing a type attribute), and the default one is the file generator. This generator generates SAX events by parsing an XML file. The transformer is again the default one, which is the XSLT transformer. The serializer then is the html serializer (which is also the default one, but I've just added the type attribute for illustrative purposes).

Something about the pattern attribute of the map:match: there can be different types of matchers, the default one is the wildcard matcher. This means that in the pattern, you can use wildcards, and then later on refer to the values of those wildcards. The following example illustrates this:

```
<map:match pattern="/pages/*/*/**">
  <map:generate src="/myrepository/{1}/documents/{2}-{3}"/>
  ...
</map:match>
```

One star in the pattern matches any character except the forward slash. Two stars match any character, including the forward slash. You can then later on refer to the matched text using the {star-position} syntax, as shown in the src attribute of the map:generate. Next to the default wildcard matcher, Daisy also uses the regexp matcher, which is basically the same but uses regular expressions to perform the matching.

5.18.1.2 Apples

Next to executing XML-transforming pipelines, Cocoon can also call controller logic. There are several implementations of the flow-engine: a javascript based one, one called "javaflow", and one called "apples". The first two feature a special thing called continuations, but since we don't use that, just ignore them. Apples sure is the least-used one (it might well be that we're the only ones using it), but its simplicity and stability make it a good choice.

So, how do these apples work?

An apple controller is a Java class that implements the interface AppleController:

```
public interface AppleController {
    void process(AppleRequest req, AppleResponse res) throws Exception;
}
```

This should speek pretty much for itself, it is a servlet-like interface. We'll go into the details further on.

In the sitemap, an apple controller can be called as follows:

```
<map:match pattern="/mypage">
  <map:call function="<name-of-the-apple-class>"/>
</map:match>
```

So when you enter the path /mypage in the browser, Cocoon will run over the sitemap and encounter the matcher matching /mypage. When it then sees the map:call, it will instantiate the class mentioned in the function attribute (the name function for this attribute is a bit weird, it is from the time the javascript flow solution was the only one). Then it will call the process(req, res) method of the apple.

Now let's look a bit more at the AppleRequest and AppleResponse objects. The AppleRequest object gives access to the Cocoon Request object, which is about the same as the Servlet Request object (Cocoon defined its own interface to be independent from servlet containers, you can also run Cocoon on the command line for example). Next to that, it offers access to sitemap parameters, which are extra parameters which can be defined using <map:parameter name="..." value= "..."/> tags inside the map:call element.

The AppleResponse object tells Cocoon what it should do after the process() method returns. There are two possibilities:

- either a Cocoon pipeline is called to render a page (in which case an object, typically a Map, can be passed to that pipeline)

- or a HTTP redirect is performed to bring the user/browser to another page

Knowing all this, you can see how this gives a web-MVC-style of working: the sitemap is used to determine a controller to call, the controller then does it works and finally lets Cocoon render a page (based on a pipeline described in the sitemap), or redirect to another URL.

There's still one important thing that we didn't mention: Apples are stateful objects. After Cocoon instantiates the Apple, it will generate a random ID and store the apple, whereby the random ID can later be used to retrieve the apple instance. This random ID is called the continuation id. How is this useful? Well, let's first look at how an existing apple instance can be called from the sitemap:

```
<map:match pattern="/mypage/*">
  <map:call continuation="{1}"/>
</map:match>
```

When you do a request for e.g. /mypage/123, the above matcher will match it, and the value "123" is passed as continuation ID to the map:call statement. This will cause Cocoon to look up the apple instance stored under the given ID, and call the process() method of the Apple. So you could say that the flow continues in the same apple instance.

The advantage of this Apple-approach is that the instance variables of the apple can store objects related to a certain flow instance. In contrast to plain servlets, you don't need to store them in the session and afterwards retrieve them from there. Also, in contrast with plain session-based solutions, you've got separate flow data for each instance of the Apple, and thus the same flow can be run multiple times in parallel even if there's only one session.

If you don't need the flow logic to be stateful, let the Apple implement the interface StatelessAppleController in addition to the AppleController interface (StatelessAppleController is only a marker interface).

5.18.1.3 First look at the Daisy sitemap

The logical structure of the Daisy sitemap (and of a sitemap in general) is as follows:

```
<map:sitemap>
  <map:components>
    ...
  </map:components>

  <map:resources>
    ...
  </map:resources>

  <map:pipelines>
    <map:pipeline internal-only="true">
      ...
    </map:pipeline>

    <map:pipeline>
      <map:parameter name="expires" value="access plus 5 hours"/>
      ...
    </map:pipeline>

    <map:pipeline>
      ...
    </map:pipeline>

  </map:pipelines>
</map:sitemap>
```

Now some explanations about all this:

Cocoon allows to have multiple sitemaps and to mount one sitemap into another, this allows to split off the handling of a part of a site to a separate sitemap. There is one root sitemap, this is the one where it all starts and this is the one located in the root of the webapp directory. Daisy ships with the default Cocoon root sitemap, which contains a lot of component definitions and automatic mounting of sitemaps located in subdirectories of the webapp directory. The Daisy sitemap itself is meant to be used as a subsitemap of this root sitemap (you could use it directly as the root sitemap if you copied over all required component definitions).

In the above given fragment, the first element inside the map:sitemap element is the map:component element. This element contains the declaration of various types of sitemap components: generators, transformers, serializers, and a couple of others.

The map:resource element contains resources definitions, these are reusable pieces of pipeline defintions.

Then the map:pipelines element contains all the statements that Cocoon will run over when processing an incoming request, i.e. the matchers, generators and so on. It is however first divided using map:pipeline (without s at the end) elements. The map:pipeline elements allow to set some options, some of which are illustrated in the above example: the first map:pipeline has an internal="true" attribute. This means that this part will only be considered for internal requests: thus not requests coming from a browser, but requests coming from for example an Apple.

The second map:pipeline element is the one which contains all the matchers for static resources, therefore we specify a special parameter there to tell Cocoon that it should tell the browser that these files won't change for the next five hours. This heavily reduces requests on the webserver, since things like images, css and js files only need to be retrieved once.

The last map:pipeline is the one containing the matcher for all public non-static requests.

So if you want to figure out how a certain URL that you see in your browser is handled by Cocoon, you open Daisy's sitemap.xmap file, and look top-to-bottom at the matchers inside the last map:pipeline element. Often, these matchers will contain a map:call statement that calls an Apple. You can then go look at the source code of that Apple to see how it handles the request, and what method it calls on the AppleResponse object. Often, this will be the sendPage(String path, Object viewData) method, which will cause Cocoon to start evaluating the sitemap to match the given path (but now it starts in the first map:pipeline element because it is an internal request).

5.18.1.4 Daisy repository client integration

If you read our introduction to Merlin, you'll remember about the Avalon Framework interfaces like Configurable, Initializable and Serviceable. Well, Apples (and all Cocoon components in general) can also implement these interfaces and Cocoon will honour them. This is because Cocoon is also Avalon-based, though it doesn't use Merlin as runtime container.

The Daisy repository client (the remote implementation of the RepositoryManager interface) is available inside Cocoon's component container, and you can get a handle at it in an Apple by implementing the Serviceable interface so that you get access to a serviceManager from which you can retrieve it. (Note that in contrast to Merlin, where each component needs to define its dependencies, the component container in Cocoon is less strict and allows you to retrieve any component).

If you look at the implementation of a Daisy Apple, you'll see that the work of getting a repository instance is done by the method LoginHelper.getRepository(request, serviceManager).

This method will first check if there is a Repository object stored in the session (which is stored there when logging in) and if so return that, otherwise it will create a default guest-user Repository object.

5.18.1.5 Actions

In the Daisy sitemap, you'll also see a couple of map:act statements. These call a so-called Cocoon action, which is basically a Java class that can do whatever it wants. In contrast with an Apple called by a map:call statement, the evaluation of the sitemap continues after encountering a map:act statement.

5.18.1.6 JX Templates

In the Daisy sitemap many pipelines start with a map:generate with type="jx" . This is the JXTemplate generator, which executes a template. This template can access the data that you passed as the second parameter in the AppleResponse.sendPage(path,viewData) method. The syntax for JXTemplates is described here[13].

5.18.1.7 Cocoon Forms (CForms)

For handling of web forms, Cocoon makes use of the CForms framework, which is documented over here[14].

CForms is a widget-oriented high-level forms framework. It is based on the concept of having a form definition (which defines the widgets that are part of a form) and a corresponding form template (to define how the widgets are placed in a HTML page). The form definition and form template are simply XML files. One handy side-effect is that for Daisy's document editor, for which the structure depends on the document type, we can simply generate the form definition and form template by performing an XSLT transform on the XML representation of a Daisy document type (see doctype_to_form.xsl and doctype_to_formtemplate.xsl)

This last sentence isn't entirely correct anymore, since the implementation of the editor has changed a bit. It is now only the fields form which is dynamically generated, using doctype_to_fieldedit_form.xsl.

5.18.1.8 Extending the Daisy Wiki application

The default Wiki application offers an extension mechanism and sample, showing you how to include outside functionality inside the Wiki application (and default look and feel), this is discussed elsewhere (page 198). Also, here[15] is a very simple example showing you how to integrate external RSS feeds into a Daisy webpage.

Notes

1. http://www.google.com/search?q=$%7B1
2. http://www.webdesignpractices.com/navigation/facets.html
3. http://cocoondev.org/main/facetedBrowser/default
4. http://www.poorbuthappy.com/fcd/
5. http://www.w3.org/TR/xsl/

6. http://xml.apache.org/fop/
7. http://cocoon.apache.org
8. http://www.ibiblio.org/maven/cocoon/jars/
9. http://www.ibiblio.org/maven/cocoon/jars/cocoon-html-2.1.7.jar
10. http://issues.cocoondev.org/
11. http://subversion.tigris.org
12. http://cocoon.apache.org/
13. http://cocoon.apache.org/2.1/userdocs/flow/jxtemplate.html
14. http://cocoon.apache.org/2.1/userdocs/forms/
15. http://blogs.cocoondev.org/stevenn/archives/002696.html

6 Book publishing

6.1 Daisy Books Overview

6.1.1 Introduction

The purpose of Daisy Books is to publish a set of aggregated Daisy documents as one whole, thus like a book or manual. Books can be published in various forms, including PDF and HTML. The HTML can be published as as one big page or chunked into multiple pages, the chunks don't need to correspond to the original document boundaries.

Technically, Daisy Books is part of (= integrated in) the Daisy Wiki, though the process of publishing a book is completely unrelated to that of publishing a Wiki page (except that both reuse the services offered by the Publisher (page 90) component).

Daisy Books offers a lot of common features required for books, including:

- Table Of Content generation
- Lists of figures and tables
- cross-references (to chapters, sections, figures or tables, in different styles)
- numbering of sections, figures and tables
- index
- footnotes

Books are published 'off-line' (in batch). This means a user will trigger the process to aggregate and format the Daisy documents into an actual book. The resulting book can then be consulted by other users (the book readers).

6.1.2 Terminology

Let us first agree on some terminology. Here we only give some very general definitions, which are described in more detail later on.

Book definition. A book definition is a Daisy document that defines the structure and content of the book (which documents to include), it defines general book metadata, and it defines default parameters for the publication of the book.

Publication process. A publication process generates an actual book (a book instance) based on a book definition, according to one ore more publication types.

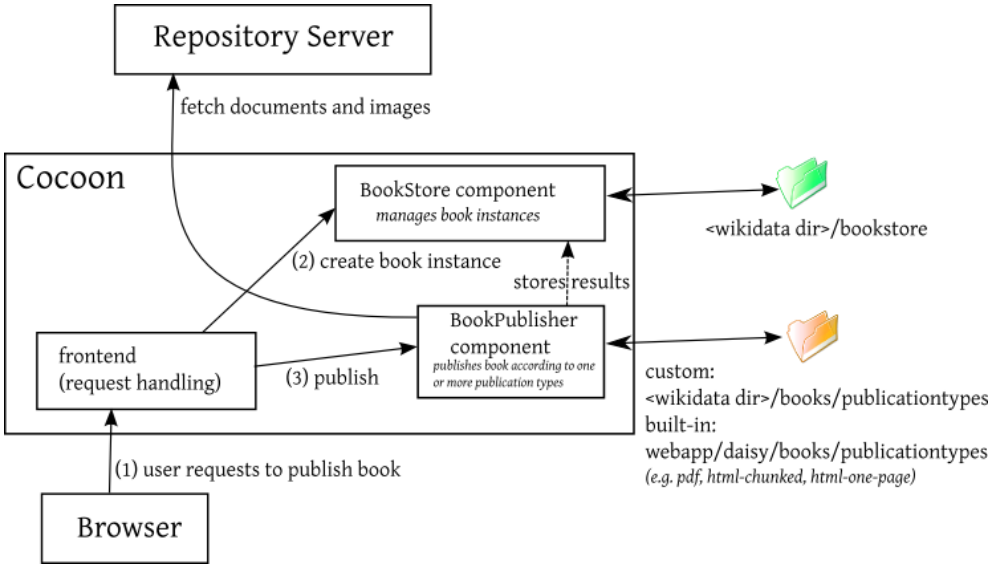
Book instance. A book instance, or simply book, is the result of running a book publication process. It is a collection of files (organised in directories) among which the published PDF file and/or HTML files, but also a snapshot of the data which is included in the book, intermediary book production files, a book publication log and a link errors log. Book instances are created and managed by the book store.

Book store. The book store manages book instances. In other words, the book store is where published books are stored.

Publication type. A publication type defines one way of how a book can be published (for example "PDF" or "Chunked HTML"). Publication types can be customised to a certain extent through properties (for example, to define the depth of the generated TOC), though for most layout changes you will usually create your own publication types (starting from the default ones).

Publication output. A publication output, or simply publication, is the result of publishing a book according to a publication type (thus a set of files).

6.1.3 The book publication process



The process of publishing a book consists of two main parts:

1. *Book data retrieval:* retrieves, from the repository server, all the data (documents and images) needed for inclusion in the book. This data is stored as part of the book instance.
2. *Publication type specific publishing:* performs the publishing according to a publication type (this step is performed multiple times if multiple publication types were selected)

These two parts are now described in some more detail.

6.1.3.1 Book data retrieval

The needed book data is retrieved and stored in the book instance. There are multiple reasons for doing this:

- all the publications (PDF, HTML, ...) will be performed on exactly the same snapshot of the book data. Otherwise, it might be that a document gets updated between the publishing of e.g. the PDF and the HTML.
- it allows to re-publish the book in the future based on the same snapshot
- it is is more efficient to retrieve the data only once

The documents are retrieved using the Publisher (page 90) component in the form of 'prepared documents'.

While retrieving the data, a list is compiled of all retrieved documents with their exact version retrieved. This information can be used later on to check what documents were updated since the publishing of a book.

6.1.3.2 Publication type specific publishing

This process is defined by the publication type, but usually follows these steps:

- apply document type specific styling on each of the documents
- shift the headers in the documents based on their hierarchical position in the book definition
- assembling the individual documents into one big XML document
- add numbers to sections, figures and tables
- add the index (only for PDFs)
- checking for double IDs, broken links, and adding links to all elements that require one (i.e. all headers, and images/tables that have a caption)
- add the table of contents and lists of figures and tables
- publication type specific steps:
    - applying a Cocoon pipeline to generate customised HTML or XSL-FO
    - chunking the HTML into multiple parts
    - ....

6.2 Creating a book

The only thing required to publish a book is a book definition.

6.2.1 Quickly trying it out

To create (publish) a book based on existing content:

- Create a book definition document: create a document as usual but choose the "Book Definition" document type. Add some section nodes to it, or alternatively use the import node to import the structure of an existing navigation tree.
- In the menu, select Tools > Book publishing. If you're logged in, you'll see a link "Publish a book".

6.2.2 Creating a new book from scratch

Usually, you will simply set up a Daisy Wiki site for the purpose of creating/editing the book content. However, instead of using a navigation tree definition, you will use a book definition.

Assuming you have no existing content yet, these are the steps to set up a new book (described in more detail below):

1. Creating a collection which will contain the documents for the books
2. Create a first document to include in the book
3. Create a book definition, for now just containing a reference to the document created in the previous step
4. Define a new Wiki site, with as home page the created document and as navigation tree the book definition.
5. Verify the site works
6. Try to publish the book

You will need Administrator access (to create the collection) and access to the machine on which the Daisy Wiki is running (to define the new site, there is no GUI for this yet).

Now lets run over these steps in more detail.

6.2.2.1 Creating a collection

In the Daisy Wiki, go to the Administration pages.

The Administration link is only visible if you have the "Administrator" role selected.

Over there, create a new collection (this should point itself out). Write down the ID of the created collection, we will need it later on.

6.2.2.2 Create a first document

In the Daisy Wiki, browse to some existing site, and create a new document (usually of type "Simple Document"). Type some initial dummy text in the document (like "hello world!"). Switch to the collections tab and add the document to the newly created collection, and remove it from any other collections.

Save the document. Write down the ID of the document (which you can see in the URL or else by selecting the "Document Info" link).

6.2.2.3 Create a book definition

Again in the Daisy Wiki, create a new document but this time of type "Book Definition". As name for the document, enter something like "First Book" or an appropriate name.

In the first part ("Book Definition Description"), click on the icon to insert a new section node, and fill in the ID of the document created in the previous step. Alternatively, if the GUI editor does not work in your browser, you can copy and paste the following XML:

```
<?xml version='1.0' ?>
<b:book  xmlns:b="http://outerx.org/daisy/1.0#bookdef">
  <b:content>
    <b:section documentId="xyz"/>
  </b:content>
</b:book>
```

Replace the xyz in the above XML with the ID of the document created in the previous step.

Then switch to the part "Book Metadata", and edit the title of the book to something appropriate.

Now save this document, and again write down its ID.

6.2.2.4 Define a new Wiki site

On the computer on which the Daisy Wiki is installed, go to the following directory:

```
<wikidata directory>/sites
```

In there, create a new subdirectory, with some name of your choice (without spaces). For example, "firstbook".

In this newly created directory, create a file called siteconf.xml with the following content:

```
<siteconf xmlns="http://outerx.org/daisy/1.0#siteconf">
  <title>First Book</title>
  <description>My first book</description>
  <skin>default</skin>
  <navigationDocId>book_def_id</navigationDocId>
  <homepageDocId>first_doc_id</homepageDocId>
  <collectionId>collection_id</collectionId>
  <contextualizedTree>true</contextualizedTree>
  <branch>main</branch>
  <language>default</language>
  <defaultDocumentType>SimpleDocument</defaultDocumentType>
  <newVersionStateDefault>publish</newVersionStateDefault>
  <locking>
    <automatic lockType='pessimistic' defaultTime='15' autoExtend='true'/>
  </locking>
</siteconf>
```

In this XML, replace the highlighted strings "book_def_id", "first_doc_id" and "collection_id" with the correct IDs of the items created in the previous steps. Then save this file.

6.2.2.5 Verify the site works

Go to the site index page (in the Daisy Wiki, the "Daisy Home" link in the right top). You should see the new site ("First Book") appear in the list. If you do not see it, wait ten seconds and refresh the page (it can take up to 10 seconds before the new site gets detected). If it does still not appear, it is usually because there is some error in the site definition. In that case, check the following log file:

```
<wikidata directory>/logs/error.log
```

Another reason a site might not appear is because you don't have read access to its home page. However, since you just created these documents, this is unlikely.

Click on the site link, you should see the usual layout with the navigation tree containing your first document.

6.2.2.6 Try to publish the book

Go to the following URL:

```
http://localhost:8888/daisy/books
```

(or equivalent according to your setup)

Choose the link "Publish a book". You get a page that shows the available book definitions. At this point, probably only the one you just created. Click on the "Publish" link next to it.

- You get a form asking some parameters, you can just press "Next" there.
- Then you get a form asking for the publication types. There is a drop down there containing the items "PDF", "HTML" and "HTML (chunked)". Choose each of them and press the add link. Then press "Next" to go to the next page.
- You get a form asking for the ACL for the published book. By default, the ACL is configured so that only you will see it. Press the button "Start book publication".
- You now see a page showing the progress of the book publishing. The book publishing happens as a background process, so you could close your browser and check back later. However, for now, just leave it as is. After a little while, the book publishing will be done and you will be presented with links to the created books. If something went wrong, you will not see those links, and you will need to look at the publication log to see what went wrong.

You can consult your published books at any later time by surfing again to the books URL:

```
http://localhost:8888/daisy/books
```

There you can also adjust the ACL so that others can see the book, or delete it.

6.2.2.7 Notes

- About the book definition and book structure:
    - The name of a document included in a book will become a section title in the book.
    - In the book definition, sections can be nested. To nested sections, header-shifting will be applied. This means that a section that is normally h1 would become a h2 if the document is nested one level deep
    - To have a section without a number, enter "anonymous" in the type field of the section (in XML mode, add the attribute type="anonymous" to its <b:section/> element).
    - Similarly, if you want to have appendixes, set the type to "appendix".

    - In the book definition you can also define sections that just insert a title, and do not correspond to a document (a bit like group nodes in navigation trees).
    - The book definition can also contain queries, like in navigation trees.
    - See also the book definition XML formats description (page 237).

- To avoid having to re-select the publication types, you can define default settings in the book definition in the part "Default book publications". See also the book definition XML formats description (page 237). The same is true for the book ACL.
- Publication types can be customised through properties. When adding the publication types in the book publication wizard, you can see the default properties and give them other values. For example, "toc.depth" might be one you might to change. You can also change the numbering patterns of the sections through these properties. See the description of the addNumbering task (page 244) for more info.
- Other things you might want to try out are:

    - assigning a caption to figures and/or tables, then you will get a list of figures or list of tables in the book (this can be disabled by modifying the "list-of-figures/tables.include-types" attributes).
    - cross references: in the document editor, assign an ID to a header, image or table. Then insert a cross-reference somewhere that points to it. (There are buttons on the toolbars for these actions).
    - index: index entries are inserted in a document by using a toolbar button.

- To modify the look of the rendered books, you can create your own publication types (page 239).

6.2.3 Converting an existing Daisy Wiki site to a book

If you have a site which you would like to publish as a book, and the navigation tree of the site represents the content of the book, you can easily transform the existing navigation tree to a book definition using an XSL, as follows:

- download the existing navigation tree XML in a file: choose the "Go to navigation document" link, right-click on the download link next to "Navigation Description", choose "Save link as" (or similar), and save the file in a directory of your choice and with a name like "navigation.xml" (or whatever).
- similarly, download this XSLT (page 0) as nav2book.xsl
- then transform the navigation.xml using the nav2book.xsl. In Linux, if you have libxslt installed, you can do this as follows:

  ```
  xsltproc nav2book.xsl navigation.xml > bookdef.xml
  ```

  There are of course various other XSLT processors you can use, sometimes they are integrated in XML editors. Or you can do the conversion manually.
- Create a new document of type "Book Definition". In the "Book Definition Description" part, copy and paste the content of the bookdef.xml file.

- In the "Book Metadata" part, copy and paste the following:

```
<metadata>
  <entry key="title">My Book</entry>
  <entry key="locale">en</entry>
</metadata>
```

  Change the title to something of your choice.
- Now save the book definition.

Optionally, you can edit the siteconf.xml of the Daisy Wiki site and change the content of the <navigationDocId> element to point directly to the book definition (the book definition can serve as a replacement of the navigation tree).

Now you can publish your book as described above under the heading "Try to publish the book".

6.3 Technical guide

6.3.1 Book Definition

This document describes the purpose and content of the various fields and parts of the BookDefinition document type.

6.3.1.1 Fields

6.3.1.1.1 BookPath

Defines a hierarchical path (separated by slashes) for display of the book on overview pages.

6.3.1.2 Parts

6.3.1.2.1 BookDefinitionDescription

The BookDefinitionDescription part defines the structure of the book.

```
<book xmlns="http://outerx.org/daisy/1.0#bookdef">
  <content>
    <section documentId="...">
      [... optionally nested sections ...]
    </section>
    [... more sections ...]
    <section documentId="..." type="appendix"/>

    <query q="select name where InCollection('book1')
              and documentType='SimpleDocument'"
           sectionType="..."
           filterVariants="true|false"/>

    <importNavigationTree id="..." branch="..." language="..." path="..."/>
  </content>
</book>
```

The <section> element can function in two ways:

- either it has a documentId attribute
- or it has a title attribute

If it has a documentId attribute, it can also have the optional attributes branch, language and version.

In both cases, the section can also have an optional "type" attribute. By default two additional types are supported:

- appendix
- anonymous (for sections that shouldn't be numbered)

Instead of listing documents explicitly, they can also be included using a query. The query element can also have an optional 'sectionType' attribute to specify the type of sections generated, and a filterVariants attribute (default true) to specify whether branch and language conditions should be added automatically.

Using the `importNavigationTree` element, it is possible to import (part of) a navigation tree and convert it to book sections. Group nodes in the navigation tree become section nodes with a title attribute, and document nodes become section nodes with a documentId attribute. The branch, language and path attributes on the importNavigationTree element are optional. Using the path attribute, it is possible to import only a part of a navigationtree. Its value should be a list of node IDs separated by slashes. If the path points to a group node, only the children of the group node are imported, if it points to a document node, the document node and its children are imported.

6.3.1.2.2 BookMetadata

```
<metadata>
  <entry key="title">Content Management</entry>
  <entry key="locale">en</entry>
</metadata>
```

The BookMetadata part contains general book metadata in the form of name-value couples. The metadata fields are accessible during the book publication.

Currently the following metadata fields are explicitly defined:

6.3.1.2.2.1 General metadata properties

Name	Purpose
title	the title of the book
locale	the locale to be used for publishing the book. This determines the language used to display items such as "Table Of Contents"

6.3.1.2.2.2 Metadata properties for configuring variables

These properties configure the variable resolving (page 220) for the book. This are book metadata properties, rather than publication properties, since the documents are retrieved only once from the Publisher (page 90) and re-used for all publications.

Name	Purpose
variables.source.X	where X is 1, 2, etc. These properties define the variable documents to be used
variables.resolve-attributes	Enables or disables substitution of variables in attributes. Should be true or false, true by default

6.3.1.2.2.3 Your own

You can add any other metadata fields of your own choice. These will be accessible during the book publishing, e.g. for use in stylesheets. If the properties are specific to one type of publication, you should publication properties instead.

6.3.1.2.3 BookPublicationsDefault

The BookPublicationsDefault part defines the publications, with their properties, that should be selected by default when publishing a book.

```
<publicationSpecs xmlns="http://outerx.org/daisy/1.0#bookpubspecs">
  <publicationSpec type="pdf" name="pdf">
    <properties>
      <entry key="toc.depth">2</entry>
      [... more entries ...]
    </properties>
  </publicationSpec>
  [... more publicationSpec elements ...]
</publicationSpecs>
```

6.3.1.2.4 BookAclDefault

The BookAclDefault part defines the default ACL to be used for the book instance created when publishing this book.

```
<bookAcl xmlns="http://outerx.org/daisy/1.0#bookstoremeta">
  <bookAclEntry subjectType="everyone" subjectValue="-1" permRead="grant"
permManage="grant"/>
  [... more bookAclEntry elements ...]
</bookAcl>
```

The subjectType attribute can have the values "user", "role" or "everyone". The subjectValue attribute should contain the ID of a user or role (corresponding to the value of the subjectType attribute), or -1 if the subjectType is everyone.

The permGrand and permManage attributes can contain "grant", "deny" or "nothing".

6.3.2 Publication Type Definition

6.3.2.1 Introduction

As already mentioned in the overview (page 230), a publication type describes how a book is published. If you want to customize this process (most often to change the look of the published books) then you can do this by creating new publication types (do not modify the built-in publication types). In this document we are going to discuss publication types in some more detail.

A publication type is defined by a subdirectory of the publicationtypes directory, which is located at:

```
For the custom publication types:
<wikidata directory>/books/publicationtypes

For the built-in publication types:
<DAISY_HOME>/daisywiki/webapp/daisy/books/publicationtypes
```

These two directories are transparently merged: first the publicationtypes directory in the wikidata directory is searched, then the one in the webapp.

The structure of the publication types directory is as follows:

```
publicationtypes/
  <name publicationtype>/
    publicationtype.xml
    sitemap.xmap
    properties.xml
    i18n/
    document-styling/
    ...
  common/
    i18n/
    properties.xml
  common-html/
    (files shared by the default HTML publications)
```

Each subdirectory of the `publicationtypes` directory which contains a `publicationtype.xml` file is considered to be a publication type. The name of the directory is the name of the publication type. Next to the `publicationtype.xml` file, the subdirectory can contain other resources such as XSLT stylesheets, a Cocoon sitemap, resources such as CSS files and images, etc.

Remember that a publication type can be customised by properties. The default properties are defined in the optional file properties.xml in the publication type directory. Properties which are common for all publication types can be defined in the common/properties.xml file. The properties used when executing the publication type are a combination of the common properties, the default properties, and the properties specified when starting the publication process.

The `common-html` directory contains some files that are shared between the two built-in HTML publications (html-one-page and html-chunked).

6.3.2.2 Creating a new publication type

To create a new publication type, simply start by duplicating one of the existing publication type directories (html-one-page, html-chunked or pdf).

When duplicating one of the HTML publication types, you will most likely also want to modify the files part of the common-html directory. Therefore, copy the contents of the common-html directory also into your new publication type directory. Then search for occurrences of the string "../common-html/" in the publicationtype.xml and sitemap.xmap files of the new publication type, and remove them. For example, a line like the following:

```
    <copyResource from="../common-html/resources" to="output/resources"/>
```

should be modified to:

```
    <copyResource from="resources" to="output/resources"/>
```

Next, edit the file publicationtype.xml and change the content of the `<label>` element to something appropriate.

When this is done, you can already try out the new publication type (which will do exactly the same as the one you copied it from).

6.3.2.3 Customizing the publication type

The publication process of a publication type is described in its publicationtype.xml file. It consists of a number of actions, which are executed one after the other. A description of these actions can be found here (page 241).

However, most often you will not need to modify that, since the layout of the books is mostly determined by the XSL stylesheets.

6.3.2.4 Publication properties and book metadata

During the book publishing, it is possible to access the publication properties and the book metadata, e.g. from the XSLTs. This way, the behaviour of a publication can be parameterised. The default publication types include examples of how to access these (or as always, just ask on the mailing list if you need more help).

6.3.2.5 Publicationtype.xml syntax

```
<publicationType xmlns="http://outerx.org/daisy/1.0#bookpubtype">
  <label>HTML</label>
  <startResource>output/index.html</startResource>
  <requiredParts partTypeNames="..."/>
  <publicationProcess>
    [... sequential list of tasks to be executed ...]
  </publicationProcess>
</publicationType>
```

The startResource is the resource that will be linked to to view the output of the publication type. This information is needed because the output of the publication process can consist of many files (e.g. for the chunked HTML publication type).

The requiredParts element is optional, and can specify for which parts the binary content should be downloaded during the dataretrieval step of the book production. By default this is only done for images. The partTypeNames attribute can contain a comma-separated list of part type names. It is also possible to implement more advanced behaviour for this, by specifying a class attribute which refers to an implementation of this interface:

```
org.outerj.daisy.books.publisher.impl.dataretrieval.PartDecider
```

In this case the partTypeNames attribute becomes irrelevant. You can add other attributes to the requiredParts element, and if your implementation of PartDecider has a constructor taking a Map as argument, the attributes will be provided via this map.

The content of the <publicationProcess> element describes the task to be performed for publishing the book, see the task reference (page 241).

6.3.3 Publication Process Tasks Reference

6.3.3.1 General

All publication tasks read and write their files in the book instance that is currently being processed. Thus all input and output paths specified on the individual publication tasks are relative paths within a book instance, and are prefixed with the directory of the current publication output, thus `/publications/<publication output name>/`.

6.3.3.2 applyDocumentTypeStyling

6.3.3.2.1 Syntax

```
<applyDocumentTypeStyling/>
```

6.3.3.2.2 Description

Applies document type specific stylesheets.

6.3.3.2.2.1 Location of the document type specific stylesheets

The stylesheets are searched in the following locations:

First a document-type-specific, publication-type-specific stylesheet is searched here:

```
<wikidata dir>/books/publicationtypes/<publication-type-name>/document-styling/<
document-type-name>.xsl
```

If not found, then a document-type-specific stylesheet is searched here:

```
<wikidata dir>/books/publicationtypes/document-styling/<document-type-name>.xsl
```

Finally, if not found, a generic stylesheet is used:

```
webapp/daisy/books/publicationtypes/common/book-document-to-html.xsl
```

6.3.3.2.2.2 Output of the document type specific stylesheets

In contrast with the Daisy Wiki, the document type specific stylesheets should always produce Daisy-HTML output. This is because later on a lot of processing still needs the logical HTML structure (such as header shifting, formatting cross references, ...). If you want to do output-medium specific things, you can leave custom attributes and elements in the output and later on interpret them in the stylesheet that will translate the HTML to its final format.

The output of the stylesheets should follow the following structure:

```
<html>
  <body>
   <h0 id="dsy<document id>"
       daisyDocument="<document id>"
       daisyBranch="<branch>"
       daisyLanguage="<language>">document name</h0>

   ... the rest of the content ...

  </body>
</html>
```

Since headers in a document start at h1, but the name of the document usually corresponds to the section title, the name of the document should be left in a `<h0>` tag. This will then later be corrected by the shiftHeaders task.

The `id` attribute on the `<h0>` element is required for proper resolving of links and cross references pointing to this document/book section.

The applyDocumentTypeStyling task writes its result files in the book instance in the following directory:

```
publications/<publication-name>/documents
```

6.3.3.3 addSectionTypes

6.3.3.3.1 Syntax

```
<addSectionTypes/
```

6.3.3.3.2 Description

In the book definition, types can be assigned to sections (e.g. a type of 'appendix' might be assigned to a section). This task will add an attribute called `daisySectionType` to the `<h0>` tag of each document for which a section type is specified.

This task should be run after the applyDocumentTypeStyling task and will make its changes to the files generated by that task (thus no new files are written).

6.3.3.4 shiftHeaders

6.3.3.4.1 Syntax

```
<shiftHeaders/>
```

6.3.3.4.2 Description

This task is deprecated, it still exists but performs no function whatsoever. Shifting headers is now performed as part of the assembleBook task.

6.3.3.5 assembleBook

6.3.3.5.1 Syntax

```
<assembleBook output="filename"/>
```

6.3.3.5.2 Description

Assembles one big XML containing all the content of the book. Thus this task combines all documents specified in the book definition in one big XML. It also inserts headers for sections in the book definition that only specify a title. If a document contains included documents, these are also merged in at the include position.

The `<html>` and `<body>` tags of the individual documents are hereby removed, the resulting assembled XML has just one `<html>` element containing one `<body>` element.

While the assembling is done, the headers in the documents are also shifted depending on their hierarchical nesting in the book definition. All headers are always shifted by at least 1, to move the h0 headers to h1, see the applyDocumentTypeStyling task.

The output is written to the specified output path (in the book instance).

6.3.3.6 addNumbering

6.3.3.6.1 Syntax

```
<addNumbering input="filename" output="filename"/>
```

6.3.3.6.2 Description

Assigns numbers to headers (sections), figures and tables.

The numbering is done based on numbering patterns specified in the publication properties. The numbering is different for sections, figures or tables of different types.

For example, the following properties define the numbering patterns for sections of the type "default":

Property name	Property value
numbering.default.h1	1
numbering.default.h2	h1.1
numbering.default.h3	h1.h2.1

The property values are numbering patterns, which define the formatting of the number, following a certain syntax, explained below.

The properties for figures and tables are called "figure.<figuretype>.numberpattern" and "table.<tabletype>.numberpattern", respectively. The numbering of figures and tables happens per chapter (per h1-level). Figures and tables are only numbered if they have a caption (defined by the `daisy-caption` attribute)

For sections, the following additional properties can be defined:

- numbering.<section-type>.increase-number: defines whether the section number must be increased when a section of this type is encountered. It can be useful to disable this for "anonymous" sections that do not require numbering, though for which the numbering of the next sections should simply continue as if the anonymous sections were not there.
- numbering.<section-type>.reset-number: defines whether the section numbering must be restarted when a section of this type is encountered after a section of another type, inside the same section level.
- numbering.<section-type>.hx.start-number: defines the initial number for sections of this type on level x (default: 1)

On the elements to which a number is assigned, the following attributes are added:

- daisyNumber: the number formatted according to the number pattern
- daisyPartialNumber: only the number of the element itself formatted according to the style indicated in the numbering pattern (1, i, I, a or A), without any of the other parts of the numbering pattern
- daisyRawNumber: the unformatted number of the element.

6.3.3.6.2.1 Syntax of the numbering patterns

Each numbering pattern should contain exactly one of the following characters: `1, i, I, a or A`. The number of the section (or figure/table) will be inserted at the location of that character. The character indicates the type of numbering (e.g. A for numbering with letters).

The number of ancestor sections can be refered using 'h1', 'h2', ... till 'h9'. The number of the highest ancestor which has a number can be referred to using 'hr' ("root header").

It is possible to retrieve text from the resource bundle of the current publication type by putting a resource bundle key between $ signs, for example $mykey$.

Any other characters used in the numbering pattern will be output as-is.

6.3.3.7 verifyIdsAndLinks

6.3.3.7.1 Syntax

```
<verifyIdsAndLinks input="filename" output="filename"/>
```

6.3.3.7.2 Description

This task does two things:

1. It does some linking related checks: it will warn for double IDs, or Daisy-links and cross-references pointing to documents or IDs not present in the book. It will also warn for images of which the source starts with "file:", which is most often caused by accident. All these warnings are written to the link log.
2. It will assign IDs to the following elements that do not have an ID yet:

    - headers
    - images and tables which have a caption

6.3.3.8 addIndex

6.3.3.8.1 Syntax

```
<addIndex input="..." output="..."/>
```

6.3.3.8.2 Description

This task generates the index based on the index entries in the document (which are marked with `<span class="indexentry">...</span>`).

It collects all index entries, sorts them, creates hierarchy in them (by splitting index entries on any colon that appears in them), and writes out the original document with index appended before the body close tag, whereby the output has the following structure:

```
<h1 id="index">Index</h1>
<index>
  <indexGroup name="A">
    <indexEntry name="A...">
```

```
      <id>...</id>
      <id>...</id>
      [... more id-s ...]
      [... nested index entries ...]
    </indexEntry>
    [... more index entries ...]
  </indexGroup>
  [... more index groups ...]
</index>
```

The <indexGroup> elements combine index entries based on their first letter. Any entries before the letter A are grouped in an <indexGroup> without a name attribute.

The <id> elements inside the indexEntries list all the IDs of the indexentry-spans that define this index entry.

Note that this task will also assign IDs to the indexentry spans.

6.3.3.9 addTocAndLists

6.3.3.9.1 Syntax

```
<addTocAndLists input="filename" output="filename"/>
```

6.3.3.9.2 Description

This task creates the Table Of Contents and the lists of figures and tables.

6.3.3.9.2.1 Table Of Contents (TOC)

The TOC is created based on the HTML header elements (h1, h2, etc.). Only headers up to a certain level are included, which is configurable using the publication property called "toc.depth", whose value should be an integer number (1, 2, etc.).

The TOC is inserted at the beginning of the document, after the <body> opening tag, and has an XML like this:

```
<toc>
  <tocEntry targetId="..." daisyNumber="..." daisyPartialNumber="..."
daisyRawNumber="...">
    <caption>...</caption>
    [... nested tocEntry elements ...]
  </tocEntry>
  [... more tocEntry elements ...]
</toc>
```

The targetId attribute is the ID of the corresponding header. The daisyNumber, daisyPartialNumber and daisyRawNumber attributes are only present if the corresponding number had a number assigned by the addNumbering task. See the description of that task for the meaning of these attributes.

The caption element contains the content of the header tag, including any mixed content. However, footnotes or index entries which might occur in the heading are not copied into the caption element.

6.3.3.9.2.2 Lists of figures and lists of tables

Lists of figures and lists of tables are created per type of figure or table. The types for which the lists should be created have to specified in two properties:

- list-of-figures.include-types
- list-of-tables.include-types

These properties should contain a comma separated list of types. For figures and tables that do not have a specific type assigned, the type is assumed to be "default". For example to have a list of all default figures, and a list of all figures with type "screenshot", one would set the list-of-figures.include-types property to "default,screenshot". Note that the order in which the types are specified is the order in which the lists will be inserted in the output.

The lists are inserted in the output after the TOC, and have an XML structure like this:

```
<list-of-figures type="...">
  <list-item targetId="..." daisyNumber="..." daisyPartialNumber="..."
daisyRawNumber="...">the caption</list-item>
  [... more list-item elements ...]
</list-of-figures>
```

For tables the root element is "list-of-tables".

6.3.3.10 applyPipeline

6.3.3.10.1 Syntax

```
<applyPipeline input="..." output="..." pipe="..."/>
```

6.3.3.10.2 Description

This task calls a Cocoon pipeline in the publication type sitemap.

The pipeline is supplied with the following parameters (flow context attributes):

- bookXmlInputStream: an inputstream for the file specified in the input attribute
- bookInstanceName
- bookInstance (the BookInstance object)
- locale: java.util.Locale object for the locale in which to publish the book
- localeAsString: the locale as a string
- pubProps: java.util.Map containing the publication properties
- bookMetadata: java.util.Map containing the book metadata
- publicationTypeName
- publicationOutputName

The pipe attribute specified the pipeline to be called (thus the path to be matched by a matcher in the sitemap). The output of the pipeline execution is saved to the file specified in the output attribute.

For practical usage examples, see the default publication types included with Daisy.

6.3.3.11 copyResource

6.3.3.11.1 Syntax

```
<copyResource from="..." to="..."/>
```

6.3.3.11.2 Description

Copies a file or directory (recursively) from the publication type to the book instance. As with all other tasks, the "to" path will automatically be prepended with the directory of the current publication output (`/publications/<publication output name>/`).

6.3.3.12 splitInChunks

6.3.3.12.1 Syntax

```
<splitInChunks input="..." output="..." firstChunkName="..."/>
```

6.3.3.12.2 Description

Groups the input into chunks. New chunks are started on each <hX>, in which X is configurable using the publication property "chunker.chunklevel".

The output will have the following format:

```
<chunks>
  <chunk name="...">
    <html>
      <body>
        [content of the chunk]
      </body>
    </html>
  </chunk>
  [... more chunk elements ...]
</chunks>
```

By default the name of each chunk will be the ID of the header where the new chunk started, except for the first chunk for which the chunk name can optionally be defined using the firstChunkName attribute on the splitInChunks task element.

The original <html> and <body> elements are discarded, the new <chunks> element will be the root of the output. New <html> and <body> elements are inserted into each chunk, so that the content of each chunk forms a stand-alone HTML document.

6.3.3.13 writeChunks

6.3.3.13.1 Syntax

```
<writeChunks input="..." outputPrefix="..." chunkFileExtension="..."
             applyPipeline="..." pipelineOutputPrefix="..."
chunkAfterPipelineFileExtension="..."/>
```

6.3.3.13.2 Description

Writes the content of individual chunks, as created by the splitInChunks taks, to separate XML files.

The attributes applyPipeline, pipelineOutputPrefix and chunkAfterPipelineFileExtension are optional. If present, the pipeline specified in the applyPipeline attribute will be applied to each of the chunks, and the result will be written to a file with the same name as the original chunk, but with the extension specified in the attribute chunkAfterPipelineFileExtension.

6.3.3.14 makePDF

6.3.3.14.1 Syntax

```
<makePDF input="..." output="..."/>
```

6.3.3.14.2 Description

Transforms an XSL-FO file to PDF. The current implementation uses the (commercial) Ibex PDF serializer. [todo: note on serialized execution]

6.3.3.15 getDocumentPart

6.3.3.15.1 Syntax

```
<getDocumentPart propertyName="..." propertyOrigin="..." partName="..." saveAs="..."
setProperty="..."/>
```

6.3.3.15.2 Description

This task retrieves the content of a part of a Daisy document. The Daisy document is specified using a "daisy:" link in a publication property or book metadata attribute.

- propertyName: specifies the name of the property
- propertyOrigin: either 'publication' for a publication property or 'metadata' for a book metadata attribute
- partName: the name of the part from which to get the data. For example, "ImageData" for images.
- saveAs: where the data should be saved.
- setProperty (optional): specifies the name of (publication) property which will be set to true if the part data has been effectively retrieved

This task can be useful when you let the user specify e.g. a logo to put in the header or footer by specifying a daisy link in a publication/metadata property.

6.3.3.16 copyBookInstanceResources

Previously (Daisy 1.4) this was called copyBookInstanceImages. This old currently name still works for backwards-compatibility.

6.3.3.16.1 Syntax

```
<copyBookInstanceResources input="..." output="..." to="..."/>
```

6.3.3.16.2 Description

Copies all resources which are linked to using the "bookinstance:" scheme to the directory specified in the to attribute, unless the resource would already be in the output directory (thus when the resource link starts with "bookinstance:output/"). The links are adjusted to the new path and the resulting XML is written to the file specified in the output attribute.

The following resource links are taken into account:

HTML element	Corresponding attribute
img	src
a	href
object	data
embed	src

This task is ideally suited to copy e.g. the images to the output directory when publishing as HTML (for PDF, this is not needed since the images are embedded inside the HTML file).

6.3.3.17 zip

6.3.3.17.1 Syntax

```
<zip/>
```

Creates a zip file containing all files in the output directory. The zip file itself is also written in the output directory, with as name the name of the book instance concatenated with a dash and the name of the publication.

6.3.3.18 custom

6.3.3.18.1 Syntax

```
<custom class="..." [... any other attributes ...]/>
```

6.3.3.18.2 Description

This task provides a hook for implementing your own tasks. A publication process task should implement the following interface:

```
org.outerj.daisy.books.publisher.impl.publicationprocess.PublicationProcessTask
```

The implementation class can have three possible constructors (availability checked in the order listed here):

- A constructor taking an XMLBeans[1] XmlObject object as argument. The XmlObject will represent the <custom> XML element. This constructor is useful if you want to access nested XML content of the <custom> element (for advanced configuration needs).
- A constructor taking a java.util.Map as argument. The Map will contain all attributes of the <custom> element.
- A default constructor (no arguments)

6.3.3.19 renderSVG

6.3.3.19.1 Syntax

This task currently has no native tag, so it should be used through the custom task capability.

```
<custom class="org.outerj.daisy.books.publisher.impl.publicationprocess.SvgRenderTask"
        input="..." output="..."/>
```

The following table lists additional optional attributes.

attribute	description
outputPrefix	where the generated SVGs should be stored in the book instance, relative to the output of the current publication process, default: `from-svg/`
format	jpg or png. default: jpg. the Ibex XSL-FO renderer doesn't seem to handle the png's.
dpi	dots per inch, default: 96. For good quality, put this to e.g. 250.
quality	for jpegs, by default 1 (should be a value between 0 and 1)
backgroundColor	for transparent areas in images, color specified in a form like #FFFFFF (default: leave transparent)
enableScripts	should scripts in the SVG be executed, default: false. (Note: the Rhino version included with Cocoon 2.1 doesn't work well together with Batik, this can be resolved by upgrading to rhino 1.6-RC2, though this also needs a recompile of Cocoon -- ask on the mailing list if help needed)
maxPrintWidth	maximum value for the generated print-width attribute, in inches. The other dimension scales proportionally. Default: 6.45
maxPrintHeight	maximum value for the generated print-height attribute, in inches. The other dimension scales proportionally. Default: 8.6

6.3.3.19.2 Description

Parses the file specified in the input attribute, and reacts on all renderSVG tags it encounters:

```
<rs:renderSVG xmlns:rs="http://outerx.org/daisy/1.0#bookSvgRenderTask"
              bookStorePath="..."/>
```

The bookStorePath attribute points to some resource in the book instance, which will be interpreted as an SVG file and rendered. The renderSVG tag is removed, and replaced with an < img> tag, with:

- a src attribute pointing to the generated image (the produced image file name is the same as the original file name, but in a different directory as defined by the outputPrefix attribute)
- height and width attributes specifying the size in pixels (for HTML)
- print-height and print-width attributes specifying the size in a form suited for XSL-FO (e.g. 3in), depending on the specified dpi.

To make use of this task, you will typically download the content of some document part in the book instance using <requiredParts/>, and have a doctype XSL which generates the renderSVG tag. This will eventually be illustrated in a tutorial on the community Wiki.

The renderSVG task requires Batik, which is (at the time of this writing) not included by default in the Daisy Wiki.

6.3.3.20 callPipeline

6.3.3.20.1 Syntax

This task currently has no native tag, so it should be used through the custom task capability.

```
<custom
class="org.outerj.daisy.books.publisher.impl.publicationprocess.CallPipelineTask"
        input="..." output="..." outputPrefix="..."/>
```

The outputPrefix attribute is optional and defaults to `after-call-pipeline/`.

6.3.3.20.2 Description

Parses the file specified in the input attribute, and reacts on all `callPipeline` tags it encounters. This is different from the applyPipeline task which applies a Cocoon pipeline on the file specified in the input attribute itself. This task is ideally suited to do some processing on part content downloaded using <requiredParts/>. The callPipeline tag is typically produced in the document-type specific XSLT for the document type containing the part.

Syntax for the callPipeline tag:

```
<cp:callPipeline xmlns:cp="http://outerx.org/daisy/1.0#bookCallPipelineTask"
                 bookStorePath="..."
                 pipe="..."
                 outputPrefix="..."
                 outputExtension="...">
   ... any nested content ...
</cp:callPipeline>
```

When such a tag is encountered, the Cocoon pipeline specified in the pipe attribute will be applied on the document specified in the bookStorePath attribute. The pipeline is a pipeline in the sitemap of the current publication type.

The outputPrefix attribute is optional, and can specify an alternative outputPrefix than the one globally configured. The outputExtension attribute is optional too, and specifies an extention for the result file. The base file name is the same as the current filename (the one specified in the bookStorePath attribute).

The cp:callPipeline tag itself is removed from the output, however its nested content is passed through. For all elements nested inside <cp:callPipeline>, the attributes will be searched for the string `{callPipelineOutput}`, which will be replaced with the path of the produced file. For example, if you want to transform some XML file to SVG and then render it with the renderSVG task, you can do something like:

```
<cp:callPipeline bookStorePath="something"
                 outputPrefix="something/"
                 pipe="MyPipe">
  <rs:renderSVG bookStorePath="{callPipelineOutput}"/>
</cp:callPipeline>
```

If you would put the above fragment in an XSL, don't forget to escape the braces by doubling them: `"{{callPipelineOutput}}"`.

6.3.4 Book Store

6.3.4.1 General

The Book Store is responsible for the persistence of published books. It simply uses the filesystem as storage area.

By default, the Book Store stores it files in the following directory:

```
<wikidata directory>/daisywiki/bookstore
```

The location of this directory is however configurable in the cocoon.xconf:

```
<storageDirectory>file:/${daisywiki.data}/bookstore</storageDirectory>
```

If you have multiple Daisy Wiki instances in front of the same Daisy repository, you may configure them to use the same bookstore directory. Thus this directory may be shared by multiple processes.

6.3.4.2 Structure of the bookstore directory

For each book instance, the bookstore directory contains a subdirectory.

```
bookstore/
   <name of bookinstance>/
      lock
      acl.xml
      metadata.xml
      publications_info.xml
      data/
         dependencies.xml
         book_definition_processed.xml
         documents/
            <docid>_<branchId>_<langId>.xml
         resources/
            <id>_<branchId>_<langId>_<version>
```

```
            <id>_<branchId>_<langId>_<version>.meta.xml
      publications/
         log.txt
         link_errors.txt
         <publication_output_name>/
            output/
```

About these files:

- all write operations require to take a lock, which creates the lock file. When a book instance is locked, nobody can access it except the user who took the lock.
- the acl.xml defines the access control list for the book instance. See below.
- the metadata.xml file contains general metadata about the book instance.
- the publications_info.xml file contains information about the publications available in the book instance.
- the data directory contains all the retrieved book data
- the publications directory contains a subdirectory for each publication output. The files which should be publicly available should be located in the output subdirectory.

6.3.4.3 Access control on book instances

The access to a book instance is controlled by an Access Control List (ACL). The ACL determines whether a user has no permissions at all, only read permissions, or manage permissions. Read permissions allows only read access to the following paths in the book instance:

- `/publications/<publication_output_name>/output/*`
- `/data/resources/*`

Manage permission allows everything, including deletion of the book instance.

A book instance ACL consists of a number of rules. Each rule defines whether a specific user, role or everyone is granted or denied read or manage permission. By default (the start situation when evaluating an ACL), everyone is denied everything. The ACL is always evaluated top to bottom, from the first to the last rule, later rules overwriting the result of earlier rules (thus the evaluation does *not* stop on the first matching rule).

6.3.4.4 Manual manipulation of the bookstore directory

It is allowed to make manual changes to the bookstore directory, including deletion and addition of book instance directories. It is thus also allowed to copy book instance directories between different book stores, if you have multiple Wikis running (in such cases, if they talk to different repositories, be aware though that the ACL might need modification).

Notes

1. http://xmlbeans.apache.org

7 Import/export

7.1 Import/export introduction

The export and import tools allow to copy (synchronize, replicate) documents between Daisy repositories that have different namespaces (page 60).

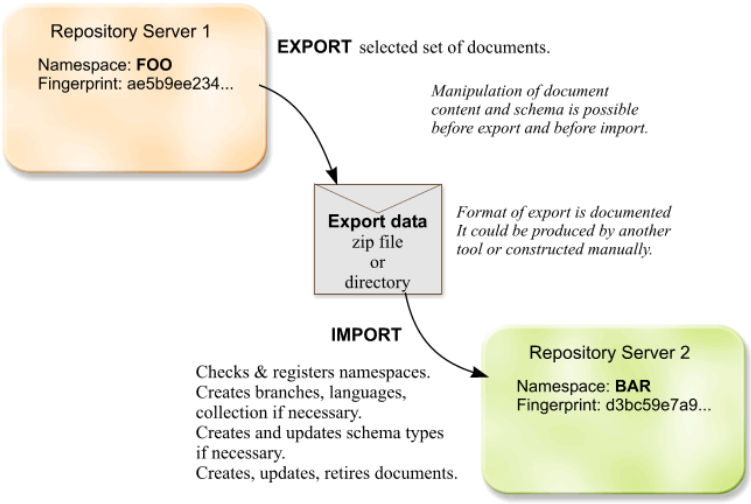
It is required that the repositories have different namespaces in order to keep the identity of the documents. For one-time transfer of documents from one repository to another, a tool could be made which recreates the documents in the target repository without preserving their IDs, which would then also require adjusting links in the content of documents. However, the import/export tools documented here do not cover this use-case.

An export not only contains documents but also schema, variant and namespace information.

The export tool can export all documents or subsets of documents (expressed using queries or a manual enumeration). Both upon export and import, manipulation of the content is possible. There are some built-in possibilities, e.g. to drop fields and parts from a document upon export or import, and for ultimate flexibility you can implement custom logic in a Java class. The import tool is smart enough to not update documents if there are no changes, so that no unncessary update events are caused.

The export and import tools are command-line applications, though the code is designed such that adding other user interfaces, or integration in for example the Daisy Wiki, should be possible.

Note that version history or time/user of the last modification is not exported/imported, the import tool simply uses the normal Daisy APIs to create/update content in the target repository. As such, the export/import tools are not a replacement for backup (page 318).



7.1.1 Applications

Export/import can be used for various purposes:

- Distributing a set of documents (for example, product documentation) to the Daisy repositories of clients, who can meanwhile also create their own content in Daisy that links to your documents. You can deliver a fresh export to them from time to time.
- Replicating content from an internal Daisy to an externally-visible Daisy. The export/import here serves as more controlled way to put documents live, and can also be used to decouple the load on the external Daisy from the internal one. In addition, you can export to multiple Daisy servers for load-balancing.
- Initializing a repository with some initial content. This is useful when doing Daisy-based projects, where you may want to upload some default data (navigation trees, home pages and other special pages). In this scenario, you will typically create the export data 'manually'.

7.1.2 Basic usage scenario

7.1.2.1 Creating an export

To define the documents to export, we need to create a file, lets call it exportset.xml, with the following content:

```
<?xml version="1.0"?>
<documents>
 <query>select id where true</query>
</documents>
```

Go to `<DAISY_HOME>/bin` (or put this directory in the PATH variable)

Execute

```
daisy-export -u <username> -p <password> -f /path/to/exportset.xml -e
/path/to/exportlocation[.zip]
```

The /path/to/exportlocation should be:

- a non-existing file ending on ".zip"

- a non-existing or empty directory

If not specified, the daisy-export tool will by default connect to a Daisy repository server on localhost listening on the default port (9263). To connect to another repository server, use the -l (lowercase L) option:

```
daisy-export -l http://hostname:9263 ....
```

Instead of specifying the password using -p, you can also leave out the -p option and you will be prompted for the password interactively.

If something goes wrong, the following message will be printed:

```
*** An error occured ***
```

followed by a description of the error and a filename to which the complete stack trace of the error has been written (if it could not be written to a file, it will be printed on screen).

If everything succeeds, a summary of the number of exported documents will be printed, as well a file will be generated with a detailed listing of these documents.

Export or import can be safely interrupted by pressing Ctrl+C. If you do this in the middle of an import or export, the summary file that lists the documents processed so far will still be created.

7.1.2.2 Importing the export

To import the just created export, execute

```
daisy-import -u <username> -p <password> -r Administrator -i
/path/to/exportlocation[.zip]
```

The -r argument specifies that the Administrator role should be made active when logging in to the repository. The Administrator role is required to create certain administrative entities such as namespaces, branches, languages, and schema types. If these would already exist, a non-Administrator role can be used as well.

Again, specify -l (repository server URL) if necessary.

If everything succeeds, a summary of the number of imported documents (and failures) will be printed, and a file will be generated with a detailed listing of the imported and failed documents.

7.2 Import tool

The import tool imports data conforming to the Daisy import/export format (page 265) into a Daisy repository. This data will usually be created with the export tool, but this is not a requirement, the data simply needs to conform to the required format and might hence be produced manually or by custom tools.

For a basic usage scenario see the introduction (page 255).

To see all command line arguments supported by the import tool, execute:

```
daisy-import -h
```

7.2.1 About versions

You should by preference use the version of the import tool corresponding with the repository server to which you are importing, otherwise it might not work. The import tool will check this is the case, though this check can be circumvented using the -s argument.

7.2.2 Administrator role

For creation or update of certain entities during the import, the Administrator role is required. This is for registering namespaces, creating branches, languages and collections, and creating or updating schema types.

If the required entities already exist, you can use a non-Administrator role as well. You might want to set the schemaCreateOnly option (see below) to true, to avoid that the import fails for non-Administrator roles if there is some slight difference between a schema type in the import and the one in the repository.

7.2.3 Importing a subset of documents

Usually the import tool will import all documents available in the export data. Sometimes it is useful to import only a subset of the documents. This is done in the same way as the set of export documents is specified for the export tool (page 262). Note that if any queries are specified in the document set file, these queries are executed on the target repository.

7.2.4 Re-try import of failed documents

If a number of documents failed to be imported for reasons which are meanwhile resolved (access control, locked documents), you can re-import just those failed documents by using the "Importing a subset of documents" technique described above. You can simply copy the list of failed documents from the import summary file to a new import-set document.

For example, in the import summary file you might find this:

```
<failedBecauseLocked>
<document id="2-BOE" branch="main" language="default"/>
</failedBecauseLocked>
```

Which you can copy into a new import set file, e.g. `importset.xml`:

```
<documents>
<document id="2-BOE" branch="main" language="default"/>
</documents>
```

It is no problem if the <document> element has extra attributes or nested content, this will be ignored.

Then you can do the import using:

```
daisy-import -f importset.xml -i originaldata.zip -u <username> -p <username>
```

This will only import the files listed in importset.xml from the originaldata.zip.

If the original import isn't too large, you might want not to bother to create such an importset.xml file and simply run the whole import again.

7.2.5 Configuration

The behaviour of the import tool can be influenced by a number of options. In addition, the documents and the schema can be altered upon import.

The configurations are done in an XML file which is supplied via the -o argument. To see an example of the such a file, which also shows the default configuration, use the -d argument:

```
daisy-import -d > importoptions.xml
```

7.2.5.1 Options

A description of the options is given in the table below.

Option	Description
validateOnSave	Sets whether documents should be validated against their document type when saving (to check that all required parts and fields are present). Default true.
documentTypeChecksEnabled	Sets whether document type checking should be performed when setting fields and parts on a document. Settings this to false allows to set fields and parts on a document which are not allowed by the document type. Default true.
createDocuments	Sets whether non-existing documents should be created. Default true. You might set this to false if you only want to update existing documents, and not create new ones. If you set this to false, you probably want to set createVariants to false too.
createVariants	Sets whether document variants should be created. Thus, if a document already exists, but not yet in the needed variant, should the variant be added? Default true.
updateDocuments	Sets whether documents should be updated if they already exist in the repository. Default true. If you set this to false, existing documents (document variants) will be left untouched.
maxSizeForDataCompare	Sets the maximum size of part data to be compared. When overwriting the content of an existing part in a document, the import tool will first compare the new and old content to see if it has changed, to avoid needless creation of document versions and duplicate storage of equal data. This setting allows to control the maximum file size (is checked for both the new and existing data) for which such compares should be done. The value is in bytes. When negative (e.g. -1), there is no limit. To never perform a compare, set it to 0. Default is -1.
storeOwner	Sets whether the owner of documents should be imported (if available in import file).
failOnNonExistingOwner	Sets whether the import should fail on a document when the owner does not exist in the target repository. Default true. If false, setting the owner is silently skipped when the owner does not exist.
createMissingCollections	Sets whether the collections to which a document belongs should be created if they do not exist. Default true. When false, the import of the

	document will fail if a collection does not exist. If you want to remove documents from collections before import (and possibly add to another collection), you can use the document import customizer.
failOnPermissionDenied	Sets whether the import should be interrupted when a document could not be created or updated because access was denied. Default false. When false, the import will simply continue with the next document (a list of failures because of permission denied is however kept and provided at the end of the import). When true, the import will stop in case of this error.
failOnLockedDocument	Sets whether the import should be interrupted when a document could not be updated because it is locked. Default false. Behaviour similar to failOnPermissionDenied.
failOnError	Sets whether the import should be interrupted when a failure occurs (another failure then permission denied or locked document). Default false. Behaviour similar to failOnPermissionDenied.
fullStrackTracesOfFailures	Sets whether full stack traces of failures should be kept (for logging in the summary file). Default false.
enableDetailOutput	Sets whether detailed output should be printed while importing documents, instead of simply a progress bar. Default false.
checkVersion	Sets whether the version of the export repository, stored in the info/meta.xml, should be taken into account. Default true.
schemaCreateOnly	Sets whether schema types should only be created, and not updated. Default false. When true, existing schema types will be left untouched.
schemaClearLocalizedData	Sets whether the localized data (labels and descriptions) of schema types should be cleared before updating. Default false. When false, the localized data is still updated, but labels and descriptions in locales which are not present in the export will not be removed.
importVersionState	Sets whether the version state should be imported. Default true. When false, new versions will always be in state 'publish' except if the saveAsDraft option is set to true.
saveAsDraft	Sets whether new versions should have the state 'draft' (rather than publish). Default false. When importVersionState is true, than that takes precedence, except if there would be no version state information in the export.
unretire	Sets whether documents should be made non-retired if they were retired in the target repository. Default true.
excludeFilesPattern	A regular expression to ignore directories from the import structure. Usually when encountering an unrecognized directory name in the document directories, the import tool will print a warning, but it can be useful to exclude certain directories from this check, such as ".svn" in case your import data sits in a Subversion repository.

7.2.5.2 DocumentImportCustomizer

A "document import customizer" allows to influence the behaviour of the document import process in a number of ways:

- the content of the document can be changed before import
- it is possible to skip the storage or removal of parts and fields
- it is possible to manipulate the repository document right before it is saved

As an example use-case, consider the image thumbnails. It makes no sense to set the part data of the ImageThumbnail part on import, since the image thumbnail is automatically generated by the repository server anyhow. In addition, it might be that the image thumbnail is not included in the import (by means of the document export customizer, in order to make the export size smaller). In that case, we don't want to sync the removal of that part either. Both cases would only cause phony updates of the documents. Therefore we would like to let the import tool simply ignore the ImageThumbnail part: don't update it, don't remove it either.

Another reason not to update or remove fields or parts is because they are local additions to the document.

The DocumentImportCustomizer is a Java interface:

```
org.outerj.daisy.tools.importexport.import_.config.DocumentImportCustomizer
```

There is a default implementation available. In addition to the use case of the ignoring parts and f ields as mentioned earlier, it also alllows to add or remove the document to/from collections.

The default import options shown by executing "daisy-import -d" show configuration examples of how to do this.

7.2.5.2.1 Creating a custom document import customizer

To create a custom implementation, you need to implement the above mentioned Java interface, and create a factory class. The factory class should have one static method called create, which takes an org.w3c.dom.Element and a Daisy Repository object as arguments. The factory class does not need to implement a particular interface. An example:

```
package mypackage;

import org.w3c.dom.Element;
import org.outerj.daisy.repository.Repository;

public class MyFactory {
  public DocumentImportCustomizer create(Element element, Repository repository) {
    ...
  }
}
```

The DOM element is the element with which the document import customizer is defined in the options XML file. It allows to read configuration data from attributes or child elements (no assumptions should be made about parents or siblings of this element).

The created custom factory class should then be mentioned in the factoryClass attribute in the options XML:

```
...
  <documentCustomizer factoryClass="mypackage.MyFactory">
...
```

In order for the import tool to find you class, it needs to be availabe on the classpath. This can be done by setting the environment variable `DAISY_CLI_CLASSPATH` before launching the import tool:

```
Unix:
export DAISY_CLI_CLASSPATH=/path/to/my/classes-or-jar

Windows:
set DAISY_CLI_CLASSPATH=c:\path\to\my\classes-or-jar
```

7.2.5.3 SchemaCustomizer

The purpose of the SchemaCustomizer is essentially the same as that of the Document Import Customizer: it allows to manipulate the schema before import, e.g. by adding or removing field or part types.

Again, there's a default implementation, which allows to drop part, field and document types. Custom implementations are possible by implementing the following interface:

```
org.outerj.daisy.tools.importexport.config.SchemaCustomizer
```

and using the same factory class mechanism as described for the document import customizer.

7.2.6 Using the import tool programatically

It is possible to embed the import tool in other applications. The import code is separate from the import tool command-line interface. The entry point to run an import is the following method:

```
org.outerj.daisy.tools.importexport.import_.Importer.run(...)
```

(This is a static method, but is no problem to run multiple imports concurrently in the same JVM).

If you want to have a look at the sources of the importer, the code is found in the Daisy source tree below `applications/importexport`.

7.3 Export tool

The export tools exports data from a Daisy repository to a directory or zip file conforming to the Daisy import/export format (page 265).

For a basic usage scenario, see the introduction (page 255).

7.3.1 Specifying the set of documents to export

The documents to export are specified in an XML file which is passed to the export tool using the -f option.

The format of this file is as follows:

```
<?xml version="1.0"?>
<documents>
    <query>select id where true</query>
    <document id="841-DSY" branch="main" language="default"/>
</documents>
```

Both the query and document elements can be used as many times as desired. The branch and language attributes on the document element are optional.

Only the documents explicitely included via the query or document elements will be part of the export. So for example images included in these documents will not be automatically added.

When using queries, if you also want to export which documents are retired, then you need to include the retired documents too, via a query option:

```
<?xml version="1.0"?>
<documents>
    <query>select id where true option include_retired = 'true'</query>
</documents>
```

7.3.1.1 Specifying extra schema types, namespaces and collections to export

By default, the export tool will export the subset of the repository schema used by the documents in the export. The same is true for namespaces and collections. If you want to force certain of these items to be part of the export, even when they are not used by the exported documents, you can also list them in the export set file, by using the following elements:

```
<documentType>...</documentType>
<fieldType>...</fieldType>
<partType>...</partType>
<namespace>...</namespace>
<collection>...</collection>
```

These elements can be mixed between the document and query elements.

It is allowed, but optional, to use the root element `exportSet` instead of `documents`.

Example:

```
<?xml version="1.0"?>
<exportSet>
    <query>select id where true</query>
    <document id="841-DSY" branch="main" language="default"/>
    <documentType>SimpleDocumentContent"/>
    <namespace>foo</namespace>
</exportSet>
```

7.3.2 Configuration

The behaviour of the export tool can be influenced by a number of options. In addition, the documents and the schema can be altered upon export.

The configurations are done in an XML file which is supplied via the -o argument. To see an example of the such a file, which also shows the default configuration, use the -d argument:

```
daisy-export -d > exportoptions.xml
```

7.3.2.1 Options

Option	Description
exportLastVersion	Sets whether the data from the last version of the document should be exported, rather than the live version. Default false.
failOnError	Sets whether the export should be interrupted when a failure occurs during exporting the document. Default false. When false, the export will simply continue with the next document (a list of failures is

	however kept and provided at the end of the import). When true, the export will stop in case of an error.
stackTracesOfFailures	Sets whether full stack traces of failures should be kept (for logging in the summary file). Default false.
includeListOfRetiredDocuments	Sets whether a list of retired documents should be included in the export. Default true.
exportDocumentOwners	Sets whether the owners of documents should be exported. Default false.
enableLinkExtraction	Sets whether link extraction should occur on document content. Default true. When true, the content of supported parts (e.g. Daisy HTML, navigation, books) are parsed for links. The branches, languages and namespaces used in those links are included in the export as non-required (the registration of these branches, languages and namespaces in the target repository is important for link extraction to work, and in case of the namespaces to make sure they are associated with the correct fingerprint). Setting this to false can save the export tool some work (the export will run a bit faster).
exportVersionState	Sets whether the state of the exported version should be excluded in the export. Default true. When the option exportLastVersion is false (the default), then the version state will always be 'publish' and thus this option serves little purpose in that case.

7.3.2.2 DocumentExportCustomizer

See the description of the document import customizer of the import tool (page 257).

The document export customizer is very similar, it is able to manipulate documents before being exported. The default implementation is able to drop fields, parts and collections, and can exclude out-of-line links and custom fields from the export. The interface to be implemented for custom implementations is:

```
org.outerj.daisy.tools.importexport.export.config.DocumentExportCustomizer
```

7.3.2.3 SchemaCustomizer

Exactly the same as for the import tool (page 257).

7.3.3 Using the export tool programatically

It is possible to embed export tool in other applications. The export code is separate from the export tool command-line interface. The entry point to run an export is the following method:

```
org.outerj.daisy.tools.importexport.export.Exporter.run(...)
```

(This is a static method, but is no problem to run multiple exports concurrently in the same JVM).

If you want to have a look at the sources of the exporter, the code is found in the Daisy source tree below `applications/importexport`.

7.4 Import/export format

This section describes the import/export format. This is the format produced by the export tool, and expected by the import tool.

The format consists of a directory structure containing files (XML files and binary data files). This directory structure can optionally be zipped, though this is not required. Both the import and export tool support zipped or expanded structures.

7.4.1 Overview

```
+ root
   + info
      + meta.xml
      + namespaces.xml
      + variants.xml
      + schema.xml
      + retired.xml
      + collections.xml
   + documents
      + <document id>
          + <branch>~<language>
               + document.xml
               + files containing binary part data
```

The import structure starts below the "root", thus there should not be a directory called "root" in the import structure.

The only file that is really required is the info/namespaces.xml file, all the rest is optional, including the documents.

7.4.2 The info/meta.xml file

This file contains some general properties.

Syntax:

```
<?xml version="1.0"?>
<meta>
  <entry key="daisy-server-version">2.0</entry>
  <entry key="export-time">2006-08-01T15:37:03.946+02:00</entry>
</meta>
```

Meaning of the properties (which are all optional):

Property	Purpose
daisy-server-version	The version number of the Daisy repository server from which the export was created. This information can be used by the import tool to do some conversions upon import.
export-time	Timestamp of when the export was created. This is added by the export tool and purely informational.

7.4.3 The info/namespaces.xml file

This file lists the namespaces that are used by the documents.

Example structure:

```
<?xml version="1.0"?>
<namespaces>
  <namespace name="FOO" fingerprint="20765a57796f4a774912606..." required="true"/>
  <namespace name="BAR" fingerprint="some finger print" required="false"/>
</namespaces>
```

Each namespace is listed with its name and fingerprint. The required attribute indicates whether the namespace is really required in order for the import to succeed. Namespaces that are really required are those of the documents themselves and those used in link-type fields. Non-required namespaces are those used in links in documents (all links outside the link-type fields). Usually the import tool will register all namespaces, unless the user running the import does not have the Administrator role. In that case, the import can continue if there are some non-required namespaces which are not registered.

7.4.4 The info/variants.xml file

This file lists the branches and languages that are used by the documents.

Example structure:

```
<?xml version="1.0"?>
<variants>
  <branches>
    <branch name="main" required="true"/>
    <branch name="foo" required="true"/>
  </branches>

  <languages>
    <language name="default" required="true"/>
    <language name="bar" required="true"/>
  </languages>
</variants>
```

The required attribute serves the same purpose as for namespaces. The description of the branches and languages can be set by adding a description attribute.

7.4.5 The info/schema.xml file

This file defines schema types (field types, part types and document types) to be imported. The root tag of this file is <schema>, and it can contain any number of <fieldType>, <partType> and <documentType> children. Field types and part types should be listed before the document types that make use of them.

So the basic structure is:

```
<?xml version="1.0"?>
<schema>
  <fieldType .../>
  <partType .../>
  <documentType .../>
</schema>
```

7.4.5.1 Common configuration

Field, part and document types can all have labels and descriptions in multiple locales. For all three, these are defined using child label and description tags. For example:

```
<documentType name="Image" deprecated="false">
  <label locale="" text="Image"/>
  <label locale="nl" text="Afbeelding"/>
  <description locale="" text="Use this document type to upload images in the Daisy
Wiki."/>
  <description locale="nl" text="Gebruik dit documenttype om afbeeldingen [...]"/>
</documentType>
```

7.4.5.2 Defining a field type

Syntax:

```
<fieldType name="..."
           valueType="string|date|..."
           multiValue="true|false"
           hierarchical="true|false"
           aclAllowed="true|false"
           allowFreeEntry="true|false"
           loadSelectionListAsync="true|false"
           deprecated="true|false"
           size="0">
  [ optional labels and descriptions ]
  [ optional selection list ]
</fieldType>
```

The name and valueType attributes are required, the rest is optional.

For a listing of the possible values for the valueType attribute, see the table later on.

There can optionally be a selection list defined. For this, use one of the following three elements.

7.4.5.2.1 Static selection lists

```
<staticSelectionList>
  <item value="...">
    <label locale="..." text="..."/>
    [ nested <item> elements in case of hierarchical list ]
 </item>
</staticSelectionList>
```

The value should be correctly formatted corresponding to the valueType of the field type. The formats are listed in a table further on.

7.4.5.2.2 Query selection lists

```
<querySelectionList query="..."
                    filterVariants="true|false"
                    sortOrder="ascending|descending|none"/>
```

7.4.5.2.3 Link query selection lists

```
<linkQuerySelectionList whereClause="..." filterVariants="true|false"/>
```

7.4.5.2.4 Hierarchical children-linked query selection lists

```
<hierarchicalQuerySelectionList whereClause="..." filterVariants="true|false">
  <linkFields>
    <linkField>[field type name]</linkField>
  </linkFields>
</hierarchicalQuerySelectionList>
```

7.4.5.2.5 Hierarchical parent-linked query selection lists

```
<parentLinkedSelectionList whereClause="..." parentLinkField="..."
filterVariants="true|false"/>
```

7.4.5.3 Defining a part type

Syntax:

```
<partType name="..."
          mimeTypes="..."
          daisyHtml="true|false"
          deprecated="true|false"
          linkExtractor="...">
  [ optional labels and descriptions ]
</partType>
```

Only the name attribute is required.

7.4.5.4 Defining a document type

Syntax:

```
<documentType name="..." deprecated="true|false">
  <partTypeUse partTypeName="..." required="true|false" editable="true|false"/>
  <fieldTypeUse fieldTypeName="..." required="true|false" editable="true|false"/>
  [ optional labels and descriptions]
</documentType>
```

The partTypeUse and fieldTypeUse elements can be used zero or more times.

7.4.6 The info/retired.xml file

This file lists documents that should be marked as retired during import.

Syntax:

```
<?xml version="1.0"?>
<retiredDocuments>
  <document id="1167-DSY" branch="main" language="default"/>
</retiredDocuments>
```

The <document> element can be repeated any number of times.

The branch and language attributes are optional and default to main and default respectively.

7.4.7 The info/collections.xml file

This file lists collections that should be created during import. Collections used in documents are not required to be listed in this file, they will be automatically created if missing. The main purpose of this file is to create additional collections.

Syntax:

```
<?xml version="1.0"?>
<collections>
  <collection>...</collection>
  [ more collection elements ]
</collections>
```

7.4.8 The documents directory

The documents directory contains all the documents to import. For each document there should be a subdirectory named after the ID of the document. This directory in turn contains again subdirectories for each variant of the document. The name structure is <branch name>~< language name>.

An example structure:

```
documents
   + 123-DSY
       + main~default
           + document.xml
       + main~nl
           + document.xml
   + 124-DSY
       + main~default
           + document.xml
```

The variant directory contains at least a document.xml file, and possibly more files for the part data (if any).

This is the minimal structure for the document.xml file:

```
<?xml version="1.0"?>
<document type="...">
  <name>...</name>
</document>
```

Everything else described below is optional.

7.4.8.1 Specifying the owner

The owner is defined by its login in an attribute called owner on the root tag:

```
<document type="..." owner="piet">
  ...
```

7.4.8.2 Specifying the version state

For the case where the import would cause a new document version to be created, the state for that version can be specified with a versionState attribute:

```
<document type="..." versionState="draft|publish">
  ...
```

7.4.8.3 Specifying fields

All fields are defined inside a fields element:

```
<document ...>
  <fields>
    <field .../>
  </fields>
</document>
```

7.4.8.3.1 Single-value fields

```
<field type="..." value="..."/>
```

Type type attribute contains the name of the field type.

The format of the values depends on the value type of the field, and is described in a table further on.

7.4.8.3.2 Multi-value fields

```
<field type="...">
  <value>...</value>
  <value>...</value>
</field>
```

7.4.8.3.3 Single-value hierarchical fields

```
<field type="...">
  <hierarchyPath>
    <value>...</value>
    <value>...</value>
  </hierarchyPath>
</field>
```

7.4.8.3.4 Multi-value hierarchical fields

Repeat the `hierarchyPath` element multiple times.

```
<field type="...">
  <hierarchyPath>
    <value>...</value>
    <value>...</value>
  </hierarchyPath>
  <hierarchyPath>
    <value>...</value>
    <value>...</value>
  </hierarchyPath>
</field>
```

7.4.8.4 Specifying parts

All parts are defined inside a parts element:

```
<document ...>
  <parts>
    <part .../>
  </parts>
</document>
```

The syntax for the part element is as follows:

```
<part type="..." mimeType="..." dataRef="..." fileName="..."/>
```

The type attribute contains the name of the part type.

The mimeType attribute specifies the mime type of the data. For example, for Daisy-HTML parts this will be "text/xml". For a PNG-image it is "image/png".

The dataRef attribute specifies the name of a file containing the data of the part. This file should be located in the same directory as this document.xml file. You are free to choose the file name.

The fileName attribute is optional and specifies the fileName property of the part (which is the file name that will be presented to the user when downloading the content of this part).

7.4.8.5 Specifying links

Example syntax:

```
<document ...>
  <links>
    <link>
      <title>...</title>
      <target>...</target>
    </link>
    [ more link elements ]
  </links>
</document>
```

7.4.8.6 Specifying custom fields

Example syntax:

```
<document ...>
  <customFields>
    <customField name="..." value="...">
    [ more customField elements ]
  </customFields>
</document>
```

7.4.8.7 Specifying collections

Example syntax:

```
<document ...>
  <collections>
    <collection>[collection name]</collection>
    [ more collection elements ]
  </collections>
</document>
```

7.4.9 Field value types and formats

valueType	format
string	as is
date	XML Schema format For example: 2006-07-14T00:00:00.000+02:00 (time component will be ignored)
datetime	XML Schema format For example: 2006-07-18T22:23:00.000+02:00 Note that maximum precision for time component supported by Daisy is seconds.
long	sequence of digits, without separators
double	sequence of digits, use dot as decimal separator
decimal	same as double
boolean	true or false
link	daisy:<docid>@<branch>:<lang> <branch> and <lang> are optional (means: same as containing document, this is recommended as the links will automatically adjust when copying data between branches and languages) <docid> is in the form <sequence number>-<namespace> Examples: daisy:123-DSY daisy:123-DSY@main:default daisy:123-DSY@:default (specifies only language, not branch)

8 Workflow

8.1 Workflow Overview

8.1.1 Introduction

The purpose of workflow is to automate business processes, in Daisy particularly those involving documents. A typical workflow process in Daisy is a document review, whereby a document might need to pass via several persons who need to give their approval.

Daisy uses a generic workflow engine, more specificially jBPM[1] (version 3.2). From a technical point of view, this workflow engine can be used to manage any sort of workflow process, even if it's completely unrelated to content management.

The structure of a workflow process is defined by a *process definition*. Such a process definition consists of nodes (of various types) with transitions between them. A specific running process (following the structure of a process definition) is called a *process instance*. Much like a normal computer program, a process instance has a pointer to where the execution is currently located (called the *token* in jBPM or the *execution path* in the Daisy API, workflow processes support multiple concurrent execution paths) and has scoped *variables*. An important difference between workflow processes and normal computer programs is that workflow processes can contain *wait states*, whereby the system needs to wait for someone (a user or another process) to perform some task. Since it might take quite some time before this task is performed, the process instances need to be persistable (storeable).

An important concept in workflow is the *task list*: task nodes in the workflow process create tasks that are assigned to users or pools of users, and appear in the task inbox of the users. The user has to perform the task and choose the transition to follow, in order to let the workflow process continue.

Workflow processes can also contain timer-triggered actions, which could e.g. be used for timed publication of documents.

For more generic information on workflow, we refer to the jBPM website[2] or any other resource[3] on workflow. The documentation here is mainly focussed on the workflow/jBPM integration in Daisy.

8.1.1.1 What Daisy does with jBPM

For those already familiar with jBPM, here's a quick summary of what the jBPM integration in Daisy involves:

- Embedded deployment in the repository server

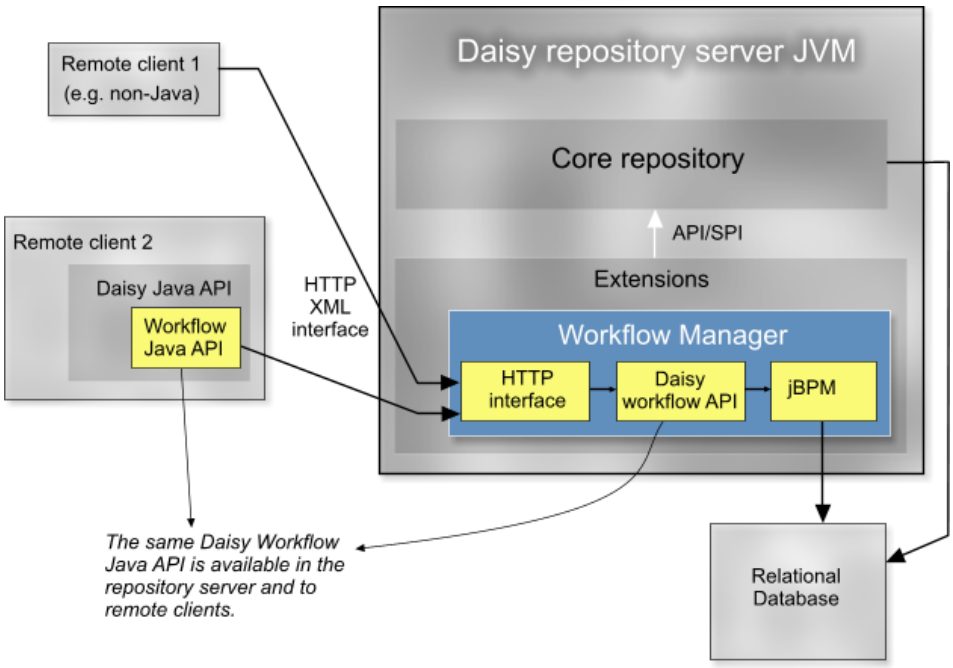
- A workflow API, with a query system
- A custom metadata layer and i18n
- Daisy Wiki-integrated GUI
- Some custom variable types to point to Daisy documents, users and actors
- Some utility classes for use in process definitions: a Javascript action, a mail action, an assignment handler

8.1.2 Workflow integration in Daisy

8.1.2.1 Workflow component in the repository server

The picture below illustrates how the workflow is implemented as an extension component in the Daisy repository server. “Extension component” means that the workflow can be disabled without any impact on the core repository server, and its code base is completely separated from the core repository, the workflow only communicates through official APIs/SPIs with the repository.

As discussed in the section on deployment (page 293), we have set up jBPM so that it stores its data in the same relational database as the Daisy repository server.



8.1.2.2 Daisy workflow API

Daisy defines its own Workflow API. This not only hides the underlying implementation, but allows to layer additional functionality (such as access control), and to provide a remote implementation.

One of the neat features of the Daisy Workflow API is its generic query system (page 288).

8.1.2.3 Daisy Wiki integration

Next to the workflow extension component in the repository server, the Daisy Wiki provides a complete workflow GUI, including:

- administration screens to deploy process definitions and manage workflow pools
- workflow console (dashboard) showing the tasks assigned to the current user and pooled tasks for the user
- the necessary screens to start workflow processes and perform workflow tasks
- search screens for processes, tasks and timers
- and the obvious little things like deleting/suspending/resuming process instances, viewing the complete details of a process instance, (re-)assign tasks, view timer details (including exception if timer failed), ...

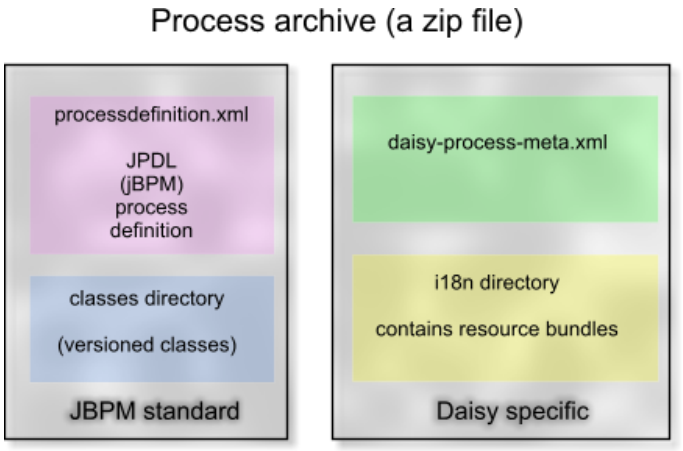
8.2 Authoring process definitions

8.2.1 Process authoring overview

8.2.1.1 Introduction

This section will get you started creating your own workflow process definitions.

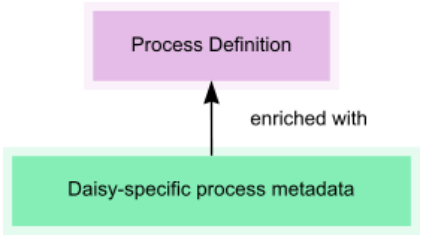
The process definitions are normal jBPM process definitions. In jBPM, a process definition is an XML file. This XML file can be deployed as-is, or can be put in a *process archive* (a zip file). The process archive can contain additional resources such as class files for process actions.



Daisy adds a custom metadata layer to the jBPM workflow process definition. This extra metadata:

- describes the task interactions (essentially comes down to a description of the variables that can be updated)
- adds localization for things like tasks and transitions (using resource bundles)

The Daisy-specific metadata is completely optional (i.e. it is possible to deploy pure jBPM process definitions), but since most tasks will require some form of input from the user (variables), you'll likely want to make use of it.



8.2.1.2 Creating process definitions: the steps

Creating a new workflow process definition involves the following steps:

1. Create a process definition file (processdefinition.xml) in the JDPL (JBoss Process Definition Language) format.
2. Create a Daisy process metadata file (daisy-process-meta.xml) adding labels and descriptions for tasks, transitions, etc, and describing the interaction with tasks.
3. Optionally create resource bundles with localized texts.
4. Zip these together
5. Deploy them via the Daisy Wiki Administration console.

8.2.1.2.1 Writing the process definition

The process definition itself is of course the most important thing. A workflow process consists of a number of nodes/states (at least a start and an end state) with transitions between them. We refer to the jBPM documentation[4] for information on how to author these. It is recommended to look not only at the chapter describing the JPDL syntax, but also to study the underlying concepts of GOP (Graph Oriented Programming).

Further in this document we'll also give more information that is relevant to authoring process definitions, such as some structural requirements imposed by Daisy (nodes, tasks and transitions should have names), and some useful utilities (writing actions in Javascript, the assignment handler).

Looking at the sample processes included with Daisy can also help you get started.

8.2.1.2.2 Creating a Daisy process metadata file

See the section daisy-process-meta.xml reference (page 280) for more information on this.

8.2.1.2.3 Creating resource bundles

See the section on internationalization (page 287) for more information on this.

8.2.1.2.4 Creating the process zip

Package the files together in a zip file, using the following structure:

```
root
  |
  +-- processdefinition.xml
  |
  +-- daisy-process-meta.xml
  |
```

```
+-- i18n (directory)
     |
     +-- messages bundle files (e.g. messages.xml, messages_fr.xml, ...)
```

8.2.1.2.5 Deploying the process definition

A process definition can be deployed using the Daisy API, or more commonly through the Administration console of the Daisy Wiki.

This last one is accessible by logging in to the Daisy Wiki, switching to the Administrator role, selecting the Administration option in the menu, and finally selecting “Manage process definitions”.

Make sure the name of your process definition (defined in the processdefinition.xml using the name attribute on the root element) doesn't conflict with the name of any existing process definition, unless of course you are deploying a new version of an existing process.

8.2.2 Workflow process examples

For examples of process definitions, we currently refer to the sources of the example processes included with Daisy. In the Daisy source tree, these can be found in this directory:

```
services/workflow/server-impl/src/processes
```

See also the latest version in SVN[5].

8.2.3 Notes on JPDL authoring

8.2.3.1 Special considerations for workflows to be deployed in Daisy

8.2.3.1.1 Process definition verification

When deploying a process definition in Daisy, Daisy applies some stricter checking on the process definition than jBPM does, more specifically:

- it checks the process definition has a name
- it checks the process definition has a start-state node
- it checks all nodes in the process definition have a name
- it checks all tasks (in task-node and the start-state task) have a name
- it checks all (leaving) transitions of the nodes have names
- it validates the Daisy metadata (if present) against its XML schema

Without these names, it is hard/impossible to refer to these entities from the Daisy-specific process metadata, or from the non-Java HTTP API, or to tell the user something about them.

If any of these checks fail, an exception is thrown and the process definition is not deployed.

8.2.3.1.2 Give all nodes, tasks an transitions names

As a result from the above mentioned verification process, all nodes, tasks and transitions in the workflow definition should have a name. The names should follow the jBPM scoping rules for uniqueness. For tasks, this means globally unique.

8.2.3.1.3 Use tasks as the only wait-states that need human interaction

The Daisy frontend GUI only supports task-based signaling (triggering) of the workflow execution. On the API level it is also possible to trigger specific execution paths, though currently there is no GUI for this.

8.2.3.1.4 Gather initial parameters using a start-state task

If you want to gather some initial parameters for the workflow process, you can do this by associating a task with the start state. In jBPM, if the task associated with the start state is associated with a swimlane (= a process role), than that swimlane gets automatically assigned to the user starting the process, hence allowing to capture the process initiator (in order to assign future tasks to the same user).

8.2.3.2 Daisy jBPM utilities

8.2.3.2.1 Using Javascript in process definitions

It is possible to script actions using Javascript in process definitions. The basic syntax is as follows:

```
<action class="org.outerj.daisy.workflow.jbpm_util.JavascriptActionHandler">
  <script>
     /* script comes here */
  </script>
</action>
```

jBPM strips and collapses whitespace and newlines from the content of the <script> element. Therefore, **do not use double-slash comments (// comment)**, since this will effectively comment out the remainder of the script.

The following variables are made available to the Javascript:

- `repository`: the repository object for the current user (see the Daisy API, Repository[6] interface). In timer actions, there is no "current user", therefore use the wfRepository instead.
- `wfRepository`: the repository object for the "workflow" user (the actual workflow user can have any name and is configured in the myconfig.xml). This is useful for multiple purposes:
  - to perform repository operations that the person who currently triggered the workflow execution isn't allowed to do (the workflow user has Administrator privileges)
  - to perform repository operations that you do not want to have associated with the current user (the user might not be aware of all the execution progress it triggers throughout the workflow)

    - in timers, which are executed asynchronously from user requests and hence have no repository object for the current user

- `repositoryManager`: the object from where Repository's can be retrieved (using a specific login/password)
- `variables`: an object providing convenient access to process variables. This object supports the following methods:

    - `variables.getGlobalVariable(name)`
    - `variables.setGlobalVariable(name, value)`
    - `variables.deleteGlobalVariable(name)`
    - `variables.getTaskVariable(name[, failOnNoTask])`: the failOnNoTask argument is optional and true by default. Causes an exception to be thrown when this method is called outside the context of a task.
    - `variables.setTaskVariable(name, value)`
    - `variables.deleteTaskVariable(name)`

- `executionContext`, `token`, `node`, `task`, `taskInstance`: these are standard jBPM objects. The task and taskInstance objects are only available if applicable.

8.2.3.2.2 Assignment handler

Daisy provides a jBPM assignment handler which performs the assignment based on the value of the process variable containing a WfActorKey[7] object. Here is an example:

```
<swimlane name="reviewer">
  <assignment class="org.outerj.daisy.workflow.jbpm_util.ActorKeyAssignmentHandler">
    <variableName>reviewer</variableName>
    <variableScope>global</variableScope>
  </assignment>
</swimlane>
```

The `<variableName>` element specifies the name of the variable, `<variableScope>` should be `global` or `task`.

A WfActorKey object can either point to a user or to one or more pools. Its fully qualified class name is:

```
org.outerj.daisy.workflow.WfActorKey
```

8.2.3.2.3 Task assignment notification mails

To send a notification mail when a task gets assigned to a user, you can associate a pre-made action with the `task-assign` event as illustrated below.

```
<task-node name="foo">
  <task name="fooTask" swimlane="initiator">
    <event type="task-assign">
      <action class="org.outerj.daisy.workflow.jbpm_util.NotifyTaskAssignAction"/>
    </event>
  </task>
  ...
```

The mail will be send to the email address of the user as configured in Daisy. If the user has no email address configured, no mail will be send (this will not give an error).

If the user to whom the task is assigned is the same as the current user (i.e. a user assigns a task to himself), no mail will be send. Similarly, when a task is unassigned, no mail will be send (as there's no-one to send the mail to).

The locale for sending the mail is currently taken from the email subscription manager, thus it is the same locale as used for sending notification mails in Daisy.

The task notification mail contains an URL to open the task. See here (page 295) for information on how to configure this URL and how to customize the mail templates.

8.2.4 daisy-process-meta.xml reference

8.2.4.1 Overview

The `daisy-process-meta.xml` contains the Daisy process metadata. This file is entirely optional, but will often be required to make meaningful usage of a workflow process in Daisy. See process authoring overview (page 275) for a general discussion on the role of the process metadata, this section only contains reference information.

One of the important things the Daisy process metadata describes is which variables can be updated as part of the interaction with each task. Any variables not declared in the Daisy process metadata cannot be accessed (read or updated) through the Daisy workflow API (though they can be used internally in the process definition).

All elements in the Daisy process metadata file are optional. The process metadata file should be valid according to its XML schema, so no additional elements or attributes are allowed (if it's not valid, deploying the process archive will fail).

The XML Schemas for the workflow can be found in the Daisy source tree in the directory `services/workflow/xmlschema-bindings/src`

8.2.4.2 Basic form

The most simple, valid process metadata file consist of only a document element, as follows:

```
<?xml version="1.0"?>
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta"
              xmlns:wf="http://outerx.org/daisy/1.0#workflow">
</workflowMeta>
```

The following sections describe the information that can be added inside this document element. The order of the elements inside the document element doesn't matter. And every one of them is optional.

As you can see, the “workflow metadata namespace” has been declared as default namespace, because most of the workflow metadata elements fall into that namespace. We've also added a declaration of the workflow namespace, since a few elements of that namespace are reused (see selectionList (page 285)).

8.2.4.3 Process label and description

These elements allow to specify a label and description for the process definition. If no label is specified, the process definition name (specified in processdefinition.xml) is used as label.

```
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta">
 <label [i18n="true"]>...</label>
 <description [i18n="true"]>...</description>
</workflowMeta>
```

The `i18n` attribute indicates whether the content of the label or description element is a resource bundle key, rather than the actual text. See workflow process internationalization (page 287) for more details on this.

8.2.4.4 Resource bundles

Zero or more resource bundles can be specified which are searched to lookup internationalized texts.

For each resource bundle, only the base name should be specified, without a directory path. The resource bundles are always retrieved from the `i18n` subdirectory in the process archive.

Having multiple resource bundles can be useful when sharing sets of messages between multiple process definitions.

For more details on resource bundles see workflow process internationalization (page 287).

```
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta">
  <resourceBundles>
    <resourceBundle>...</resourceBundle>
    [... more resource bundles ...]
  </resourceBundles>
</workflowMeta>
```

8.2.4.5 Reusable variables declarations

As detailed in the section on variables (page 282), for each task in a process it is possible to specify some variables that can be communicated to the user.

Often the same variable will be re-used by multiple tasks, therefore it is possible to declare reusable variables.

Variables can belong to the global scope or a task specific scope. For the global scope, it is mandatory to declare the variable here, in order to avoid conflicting variable definitions.

See variables (page 282) for the variable-declaration syntax.

```
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta">
  <variables>
    [... some variable declarations ...]
  </variables>
</workflowMeta>
```

8.2.4.6 Nodes

For each node in the process definition, some metadata can be specified. Currently this is only the labels for the transitions of each node.

```
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta">
  <nodes>
    <node path="...">
      <transition name="...">
        <label [i18n="true"]>...</label>
      </transition>
      [... more transitions ...]
    </node>
    [... more nodes ...]
  </nodes>
</workflowMeta>
```

The path attribute of the node element contains the fully qualified node name, which is:

```
parent node name + "/" + node name
```

For the common case of a normal top-level node (such as the start node), this means just the node name.

8.2.4.7 Tasks

For each task in the process definition, some metadata can be specified:

- a label and description for each task
- the variables that can be updated through this task

For the syntax for declaring variables inside the `<variables>` element, see variables (page 282), this syntax is the same as for the reusable variable declarations.

```
<workflowMeta xmlns="http://outerx.org/daisy/1.0#workflowmeta">
  <tasks>
    <task name="...">
      <label [i18n="true"]>...</label>
      <description [i18n="true"]>...</description>
      <variables>
        [... variable declarations ...]
      </variables>
    </task>
    [... more tasks ...]
  </tasks>

</workflowMeta>
```

8.2.4.8 Variables

Process variables allow to store state information in process instances. Together with the execution path(s), they are what make one process instance different from another.

In the Daisy Workflow API, variables are updated as part of task interactions. Only the variables declared in the Daisy process metadata are accessible, it is not possible to access other arbitrary variables.

The syntax for declaring a variable is shown below. Everything except for the `name` and `type` attributes is optional.

```
<variable name="..."
          type="string|daisy-link|long|date|datetime|actor|boolean|user"
          scope="task|global"  [optional, default task]
          required="true|false”  [optional, default false]
          readOnly="true|false" [optional, default false]
```

```
          hidden="true|false" [optional, default false]
          base="[name of a reusable variable]">
  <label [i18n="true"]>...</label>
  <description [i18n="true"]>...</description>
  <selectionList>
    [ see details below ]
  </selectionList>
  <initialValueScript>
    [ see details below ]
  </initialValueScript>
  <styling>
    [ see details below ]
  </styling>
</variable>
```

8.2.4.8.1 type attribute

The following table lists the types of process variables supported by Daisy. This means these are the types of variables which can be updated through the Daisy API, and are supported in the Daisy Wiki frontend. Inside the process definitions, variables can be assigned to any type supported by jBPM.

Name	value Java class
string	java.lang.String
daisy-link	org.outerj.daisy.workflow.WfVersionKey[8]
long	java.lang.Long
date	java.util.Date
datetime	java.util.Date
actor	org.outerj.daisy.workflow.WfActorKey
boolean	java.lang.Boolean
user	org.outerj.daisy.workflow.WfUserKey[9]

Below some more information on the Daisy-specific ones

8.2.4.8.1.1 daisy-link

This type of variable points to a Daisy document variant (identified by the triple {documentId, branch, language}). It can optionally point to a specific version of the document variant (identified by a version number or the special values 'live' or 'last').

8.2.4.8.1.2 actor

This type of variable contains an “actor”, which is either a specific user or one or more workflow pools. This type of variable is useful to use in the assignment of tasks (see assignment handler (page 279)).

8.2.4.8.1.3 user

This type of variable points to a specific user.

8.2.4.8.2 Reserved Daisy variables

Daisy defines some preserved context variables which have a special meaning. Their names all begin with "daisy_". The "daisy_" prefix should not be used for naming your own variables.

Variable name	scope	type
daisy_document	global	WfVersionKey
daisy_creator	global	WfUserKey
daisy_owner	global	WfUserKey
daisy_description	global	string
daisy_site_hint	global	string

8.2.4.8.2.1 daisy_document

While it is possible to define as many variables pointing to documents as you desire, the one called “daisy_document” is used to store the primary document with which the workflow is associated.

This variable has influence on the workflow access control (page 292).

In the Daisy Wiki, this variable is automatically initialized with the document that was active when starting the workflow.

8.2.4.8.2.2 daisy_description

This variable is used as a description of the workflow process, its value is shown in the various overview lists in the Daisy Wiki.

8.2.4.8.2.3 daisy_owner and daisy_creator

These variables gets automatically initialized by Daisy (with as value the current user) when a process instance is created. The daisy_creator is simply used to remember who created the workflow, and should never be changed. The daisy_owner variable indicates the owner of the process, which is a user who has some additional privileges (see workflow access control (page 292)). The daisy_owner may be changed after process creation.

8.2.4.8.2.4 daisy_site_hint

If this variable is present on the start state task, the Daisy Wiki will automatically put the name of the current site in it. This is useful in the task notification mails, to add an URL containing the original site name. This variable is usually declared as hidden variable.

8.2.4.8.3 scope attribute

In jBPM, variables can be bound to the tokens (the root token or other ones -- note that 'token' is called 'execution path' in the Daisy API) and to tasks. It is possible to have variables with the same name but bound to different scopes.

In the Daisy Workflow API, two variable scopes are supported:

- the “global” scope: associates variables with the root token
- the “task” scope: associates variables with specific tasks

Variables in with global scope should always be declared as part of the reusable variable declarations (page 281) and than reused in specific tasks, this to avoid conflicts in types.

8.2.4.8.4 base attribute

The base attribute is used to refer to a reusable variable declaration. When using the base attribute, usage of any other attributes is optional. So one can write:

```
<variable base="myVar"/>
```

It is however possible to customize the reused variable. The attributes and child elements can be overriden simply by specifying them. For example suppose the reusable variable is non-required, than it can be made required like this:

```
<variable base="myVar" required="true"/>
```

8.2.4.8.5 hidden attribute

Through the Daisy workflow API, it is only possible to update variables that are declared in the metadata. Sometimes you might which to have a variable programmatically available, without it being visible to the user. In that case the hidden attribute can be used.

It is recommended to make hidden variables non-required (which is the case by default), since the user is not able to enter a value for them.

8.2.4.8.6 selectionList element

A selection list allows the user to choose a value from a list instead of having to type it. The syntax for defining a selection list is as follows:

```
<selectionList>
  <listItem>
   [just one of the following, depending on the
    variable type]
    <wf:string>...</wf:string>
    <wf:date>...</wf:date>
    <wf:dateTime>...</wf:dateTime>
    <wf:long>...</wf:long>

    [an optional label]
    <label [i18n="true"]>...</label>
  </listItem>
  [... more list items ...]
</selectionList>
```

Note that the value-tags (wf:string, wf:date, etc) are in the “wf” (workflow) namespace, rather than in the workflow metadata namespace. This is because the definition of these elements is shared with other parts of the workflow XML Schema.

As an example, here is (a part of) the selection list of the priority field:

```
<selectionList>
  <listItem>
    <wf:long>4</wf:long>
    <label i18n="true">variable.priority.low</label>
  </listItem>
  <listItem>
    <wf:long>3</wf:long>
    <label i18n="true">variable.priority.normal</label>
  </listItem>
```

```
  <listItem>
    <wf:long>2</wf:long>
    <label i18n="true">variable.priority.high</label>
  </listItem>
</selectionList>
```

8.2.4.8.7 initialValueScript element

The `initialValueScript` element can contain a script that returns a default value for the variable. This is only relevant for variables associated with the start-state task, since the workflow is then not started yet (defaulting of variables for further tasks can be done inside the process definition).

The script should be written in Javascript and return a value of a type corresponding to the type of this variable. The script has access to the following variables:

- `contextDocKey`: if the workflow is started in the context of a certain document, this variable points to a WfVersionKey[10] object specifying that document (this is the same document that will be put in the daisy_document (page 284) variable).
- `repository`: the Repository[11] object for the current user
- `wfRepository`: the Repository[12] object for the special workflow user (= a repository with Administrator rights)
- `repositoryManager`: the RepositoryManager[13], allowing to retrieve any Repository

In case you simply want to initialize a variable with a fixed value, the script can contain just a return statement, for example:

```
<initialValueScript>
 return "default value";
</initialValueScript>
```

Initial value scripts however allow to do more interesting things. Here is an example for the “priority” variable that changes the priority according to the document type.

```
<initialValueScript>
  var priority = new java.lang.Long(3); // 3 is normal priority
  if (contextDocKey != null) {
    try {
      var doc = repository.getDocument(contextDocKey.getVariantKey(), false);
      var docTypeName = repository.getRepositorySchema().
             getDocumentTypeById(doc.getDocumentTypeId(), false).getName();
      if (docTypeName == "News") {
        priority = new java.lang.Long(1); // 1 is highest priority
      }
    } catch (e) {
      // something failed (e.g. no access to document), ignore
    }
  }
  return priority;
</initialValueScript>
```

Here is an example of initializing an actor-type variable. This example initializes it to contain all pools.

```
<initialValueScript>
  var pools = repository.getExtension("WorkflowManager").getPoolManager().getPools();
  var poolIds = new java.util.ArrayList(pools.size());
  for (var i = 0; i &lt; pools.size(); i++) {
    poolIds.add(new java.lang.Long(pools.get(i).getId()));
  }
```

```
  return new Packages.org.outerj.daisy.workflow.WfActorKey(poolIds);
</initialValueScript>
```

8.2.4.8.8 styling element

The styling element allows to specify styling hints for the frontend. The exact available styling hints and their meaning depends on the frontend, therefore the schema allows any attributes and child content on the styling element, so that it can be flexibly extended.

Currently supported by the Daisy Wiki frontend are:

- attribute `width`: size of the input field (e.g. `10em`)
- attribute `rows`: to enable multi-line input (textarea in HTML), its value is the number of rows you want displayed.

Example to display a variable as a textarea:

```
<styling width="10em" rows="3"/>
```

8.2.5 Workflow process internationalization

All `<label>` and `<description>` elements in the Daisy process metadata can contain either a fixed label or description or a key which will be used to lookup the label in a resource bundle. If a key is intended, add an attribute i18n="true" on the label or description element, for example:

```
<label i18n="true">variable.document.label</label>
```

i18n is short for "internationalization" (because there are 18 characters left out between the i and the n)

The resource bundles should be placed in the i18n subdirectory of the process archive. Their naming structure is as follows:

```
<basename>_<language>_<country>_<variant>.xml
```

The language, country and variant are all optional. These are some examples of valid names for the basename "messages":

```
messages_nl_BE.xml
messages_nl.xml
messages.xml
```

When a value needs to be retrieved from a resource bundle, Daisy will search all resource bundles in the order as listed in the `<resourceBundles>` tag in the daisy-process-meta.xml (page 280) file, performing fallback between languages.

So if the user's locale is `nl_BE`, Daisy will first look in `messages_nl_BE.xml`, if this file doesn't exist or the key is not found there then it will look in `messages_nl.xml`, and finally in `messages.xml`. Then the next set of bundles will be searched (if any). The purpose of multiple sets of bundles is to easily reuse sets of keys in different process archives.

Instead of the standard Java property file format, we have opted for an XML-based resource bundle format since this is more familiar and provides more comfortable encoding support. The

format used is the same as Apache Cocoon's bundles (used in the Daisy Wiki), which is something like this:

```
<catalogue>
  <message key="...">...</message>
  [... more message elements ...]
</catalogue>
```

The `<message>` elements can contain "mixed content" (= a mix of text and other elements), such as:

```
<message key="my.description">This is an <b>important</b> thing.</message>
```

8.3 Workflow query system

8.3.1 Overview

Daisy's workflow API provides a query system that allows for flexible querying of process instances, tasks and timers.

The query system allows:

- to specify a flat set of conditions on built-in properties, process variables, and task variables. A few 'special conditions' are also available. Conditions are either AND-ed or OR-ed together ('match all' or 'match any').
- results can be sorted on properties and variables.
- either the full process or task objects are returned, or a resultset (table) structure of selected properties and variables.
- the returned results can be limitted to a certain 'chunk'.

The query is not expressed in a query language but either as an object structure in the Java interface or an XML structure in the HTTP interface.

8.3.2 Example

Here is an example query expressed as XML, to be submitted via the HTTP API:

```
<?xml version="1.0"?>
<query xmlns="http://outerx.org/daisy/1.0#workflow">

  <selectClause>
    <select name="process.id" type="property"/>
    <select name="process.definitionLabel" type="property"/>
    <select name="daisy_description" type="process_variable"/>
  </selectClause>

  <conditions meetAllCriteria="true">
    <propertyCondition name="process.end"
                       operator="is_null"
                       valueType="datetime"/>
  </conditions>

  <orderByClause>
    <orderBy name="process.id" type="property" sortOrder="descending"/>
  </orderByClause>
```

```
</query>
```

You can try it out using a tool such as wget. Save the query to a file (e.g. querytest.xml), and then post it to the repository using:

```
wget --post-file="querytest.xml"
     --http-user=testuser
     --http-password=testuser
     --header="Content-Type: text/xml"
     "http://localhost:9263/workflow/query/process?locale=en-US"
```

(everything should be on one line, adjust username, password, repository URL as necessary)

8.3.3 Syntax

This is a reference of the query XML syntax. Note that this XML doesn't mention whether it is a query on tasks or process instances, this is determined by the URL to which the XML is posted.

```
<query xmlns="http://outerx.org/daisy/1.0#workflow"
       chunkOffset="[1-based offset]" chunkLength="...">

  [ selectClause is optional, if not present full entities
    will be returned ]
  <selectClause>
    <select name="[name of built-in prop or a variable"
            type="property|task_variable|process_variable"/>
    [ ...more select elements... ]
  </selectClause>

  <conditions meetAllCriteria="true|false">
    [zero or more of the following condition-describing elements]

    <propertyCondition name="[see tables of properties]"
                      operator="[see table of operators]"
                      valueType="[see tables of properties]">
      [one or more of the following elements, corresponding
       to the valueType and the number of operands the
       operator takes]
      <string>...</string>
      <date>[date in XML Schema format]</date>
      <dateTime>[dateTime in XML Schema format]</dateTime>
      <long>...</long>
      <daisyLink documentId="..." branchId="..." languageId="..." [version="..."]/>
      <actor id="..." pool="true|false">
        [in case of pools, a list of nested id elements can be used]
        <id>...</id>
      </actor>
      <user>[numeric user id]</user>
      <boolean>true|false</boolean>
    </propertyCondition>

    <taskVariableCondition [similar to property conditions] />

    <processVariableCondition [similar to property conditions] />

    <specialCondition name="[see list of special conditions">
      [any arguments the special condition takes, the value
       tags should contain type-specific tags as described
       for propertyCondition, e.g. <value><string>...</string></value>]
      <value>...</value>
    </specialCondition>
  </conditions>

  <orderByClause>
    [zero or more orderBy elements, these are very similar
     to the selectClause but with the added possibility to
```

```
     specify the sort order]
    <orderBy name="..."
             type="property|task_variable|process_variable"
             sortOrder="ascending|descending"/>
  </orderByClause>
</query>
```

8.3.4 Query reference tables

8.3.4.1 Built-in process properties

These can be used when searching for processes, tasks or timers.

Property name	Data type	Non-query
process.id	id	
process.start	datetime	
process.end	datetime	
process.suspended	boolean	
process.definitionName	string	
process.definitionVersion	long	
process.definitionLabel	(xml)	X
process.definitionDescription	(xml)	X

8.3.4.2 Built-in task properties

These can only be used when searching for tasks.

Property name	Data type	Non-query	Comment
task.id	id		
task.actor	user		
task.description	string		Currently not actively used in Daisy, see the daisy_description (page 284) process variable for a process-level description.
task.create	datetime		
task.dueDate	datetime		
task.isOpen	boolean		
task.priority	long		
task.definitionLabel	(xml)	X	
task.definitionDescription	(xml)	X	
task.hasPools	boolean	X	
task.hasSwimlane	boolean		

8.3.4.3 Built-in timer properties

These can only be used when searching for timers.

Property name	Data type	Non-query	Comment
timer.id	id		
timer.name	string		
timer.dueDate	datetime		
timer.suspended	boolean		
timer.exception	string		
timer.failed	boolean	X	To search failed timers, use "timer.exception is not null"

8.3.4.4 Operators

Description	Name	Argument count	Supported datatypes								
			id	string	daisy-link	long	date	datetime	user	boolean	actor
Equals	eq	1	X	X		X	X	X	X	X	X
Less than	lt	1		X		X	X	X			
Greater than	gt	1		X		X	X	X			
Less than or equal	lt_eq	1		X		X	X	X			
Greater than or equal	gt_eq	1		X		X	X	X			
Between	between	2		X		X	X	X			
Is null	is_null	0		X	X	X	X	X	X	X	X
Is not null	is_not_null	0		X	X	X	X	X	X	X	X
Like	like			X	X						

8.3.4.5 Data types

Name	Java class
id	java.lang.String
string	java.lang.String

daisy-link	org.outerj.daisy.workflow.WfVersionKey
long	java.lang.Long
date	java.util.Date
datetime	java.util.Date
actor	org.outerj.daisy.workflow.WfActorKey
user	org.outerj.daisy.workflow.WfUserKey
boolean	java.lang.Boolean

8.4 Workflow pools

Tasks in jBPM can be assigned to an actor, which can either be a specific user or one or more pools. The definition of the pools and who belongs to them is managed by Daisy (rather than jBPM). The Workflow API exposes methods to manage the pools.

In the Daisy Wiki, the pools can be managed through the Administration console.

8.5 Workflow access control

The workflow access control rules define what workflow operations a user can perform, in other words it is about the authorization of various workflow operations.

Currently the access control rules are not configurable. In the current implementation, there is already an interface (WorkflowAuthorizer) which could be replaced by a custom implementation, however there is no way yet to register such custom implementation.

These are the current workflow authorization rules:

- process definitions:
  - deploying, deleting, getting instance counts: users in Administrator role
  - read access: everyone
- starting a new process instance: everyone
- read access to a process instance:
  - users in Administrator role
  - the process owner
  - if the process is associated with a document using the daisy_document process variable, and the user has read access on the document
  - the user is assigned as actor for a task in the process
  - the user belongs to a pool for a task which is associated with a pooled actor. In case the process is associated with a document, the user should also have read access on the document.

    - in all other cases, no access

- update/end a task:
    - users in the Administrator role
    - the process owner
    - the actor for the task
- assign a pooled task to oneself:
    - users in the Administrator role
    - the process owner
    - users belonging to one of the pools, except if the process is associated with a document through the daisy_document process variable and the user has no read permission on the document.
- assign a task to an arbitrary actor:
    - users in the Administrator role
    - the process owner
- unassign task:
    - users in the Administrator role
    - the process owner
    - the task actor (only if the task will fall back to pooled actors)
- delete, suspend or resume a process:
    - users in the Administrator role
    - the process owner

Task and timers are only accessible if one has read access to the process to which they belong.

The results of workflow queries (page 288) are automatically filtered according to these access rules.

8.6 Workflow deployment

This section describes some deployment and management topics concerning workflow.

8.6.1 jBPM persistence (database tables)

jBPM stores all its persistent state in database tables. These tables are installed in the same database as those of the Daisy repository server. This avoids the work of setting up an additional database, and means no extra effort is required for backup.

The database tables are automatically created when the repository server starts up for the first time. After creation, this fact is recorded in the `daisy_system` table by adding an entry named `jbpm_schema_version` containing the jBPM version number. Since creating the schema can take quite some time, a warn message is printed to the log before and after doing it, and also to standard out. To force the recreation of the jBPM tables, drop all tables whose name starts with "JBPM_" and execute this statement on the database:

```
delete from daisy_system where name = 'jbpm_schema_version'
```

jBPM uses hibernate for persistence. When using MySQL, the created tables are InnoDB tables (this is achieved by configuring an appropriate hibernate dialect), which is important to guarantee consistency of the tables. The database connections for hibernate are taken from the same connection pool as used by the rest of Daisy.

8.6.1.1 Deployment of default/sample process definitions

When the jBPM tables are first created, Daisy will also deploy some default (or sample, if you wish) processes.

The source for the samples can be found in the Daisy source tree at:

```
services/workflow/server-impl/src/processes
```

or in the .zip files embedded inside `daisy-workflow-server-impl-<version>.jar`.

It is possible to trigger re-installation of the sample workflows through the Admistration console of the Daisy Wiki (and hence through the Daisy Work flow API). This can be useful when installing a new Daisy version.

8.6.2 The “workflow” user

When installing Daisy, a special user called “workflow” is created. This user can be used to access the Daisy repository in process definitions, i.e. for when processes need to retrieve or update information in the repository. This user is required to have the Administrator role. The user to be used as workflow user is configured in the myconfig.xml.

8.6.3 When workflow is not available

If you see an error like the following:

```
Received exception from repository server.
Extension named "WorkflowManager" is not available.
```

it is likely caused by the fact that the workflow user (see previous section) is not configured correctly.

8.6.4 Notification mails

8.6.4.1 Task URL

In the task assignment notification mails, an URL to the task is included. This URL is configured in the `myconfig.xml` (which is located in `<daisy data dir>/conf`) and is by default the following:

```
<taskURL>http://localhost:8888/daisy/${site}/workflow/performtask?taskId=${taskId}<
/taskURL>
```

The variable `${site}` contains the value of the daisy_site_hint (page 284) process variable, if it is available.

8.6.4.2 Mail templates

The mechanism to load mail templates uses a fallback system allowing a mail template to be loaded from different locations. The default mail templates are included inside the workflow implementation jar, but it is possible to replace this with custom versions. Mail templates are also locale-dependent, using a fallback system similar to the Java resource bundles. The mail template language is Freemarker[14].

The locations for loading mail templates are configured in the myconfig.xml, the default configuration is as follows:

```
<mailTemplates>
  <!-- built-in templates (loaded from the classpath) -->
  <location>resource:/org/outerj/daisy/workflow/serverimpl/mailtemplates/</location>
  <!-- custom templates in the datadir -->
  <location>${daisy.datadir}/templates/workflow/</location>
</mailTemplates>
```

8.6.5 Process cleanup

Daisy itself does not perform any cleanup of finished or stalled processes. Everything is kept forever until explicitly deleted.

8.7 Workflow Java API

The reference documentation of the Daisy workflow Java API is available as javadocs[15], see the following package:

```
org.outerj.daisy.workflow
```

Using the workflow API it is simple to automate everything workflow-related. In contrast to the API of the core repository server, the workflow API is "flatter", that is, there is one service interface (WorkflowManager[16]) through which all operations are performed. The returned objects are pure value objects without active behavior.

As an example, here is a bit of code I used to fill up the workflow system with some process instances for testing purposes.

This code is written as Javascript, see the instructions (page 116) on how to run this.

```
importPackage(Packages.org.outerj.daisy.repository);
importPackage(Packages.org.outerj.daisy.workflow);
```

```
importClass(Packages.org.outerj.daisy.repository.clientimpl.RemoteRepositoryManager);

var repositoryManager = new RemoteRepositoryManager("http://localhost:9263",
                                                     new Credentials("guest", "guest"));

// Add the WorkflowManager extension
repositoryManager.registerExtension("WorkflowManager",
    new Packages.org.outerj.daisy.workflow.clientimpl.RemoteWorkflowManagerProvider());

var repository = repositoryManager.getRepository(new Credentials("testuser",
"testuser"));

var wfManager = repository.getExtension("WorkflowManager");

var locale = java.util.Locale.US;
var userId = repository.getUserManager().getPublicUserInfo("testuser").getId();
var processDef = wfManager.getLatestProcessDefinition("review", locale);

var taskUpdateData = new TaskUpdateData();

taskUpdateData.setVariable("daisy_document", VariableScope.GLOBAL,
                           WfValueType.DAISY_LINK, new WfVersionKey("1-DSY", 1, 1,
"last"));

taskUpdateData.setVariable("reviewer", VariableScope.GLOBAL,
                           WfValueType.ACTOR, new WfActorKey(userId));

taskUpdateData.setVariable("taskPriority", VariableScope.GLOBAL,
                           WfValueType.LONG, new java.lang.Long(3)); // 3 is 'normal'

for (var i = 0; i < 100; i++) {
  java.lang.System.out.println(i);

  taskUpdateData.setVariable("daisy_description", VariableScope.GLOBAL,
                             WfValueType.STRING, "review process " + i);

  wfManager.startProcess(processDef.getId(), taskUpdateData,
                         null /* default transition */, locale);
}
```

8.8 Workflow HTTP interface

Here we'll look into the specifics of the workflow HTTP interface. For a general introduction, see the repository HTTP API (page 119).

Most workflow related URLs which return some XML take a "locale" request parameter to specify the prefered locale for localized content (such as labels and descriptions). For example, this can be "?locale=fr" or "?locale=en-US"

/workflow/task

GET:

- no request parameter: all tasks assigned to the current user
- with request parameter select=pooled, all tasks belonging to the user's pools

/workflow/task/<taskId>

GET: retrieves XML representation of the task

POST: depending on the value of the "action" request parameter:

- no action parameter: updates or ends task, as specified in the XML payload (wf:taskUpdateData, see workflowmsg.xsd)
- action=requestPooledTask: assign a pool task to the current user
- action=unassignTask: unassigns the task from the current actor, putting it back into the pool(s)
- action=assignTask: assign the task to a specified actor. To specify the actor, add a request parameter actorType with value "user" or "pools". The ID of the actor itself (thus the user or pool ID) is specified in a request parameter named actor. In case of pools, multiple actor parameters can be given.

/workflow/process

POST: start a new workflow process. The parameters are specified in the XML payload, which is an wf:startProcess element (see workflowmsg.xsd)

/workflow/process/<processId>

GET: retrieves an XML representation of the process instance

POST: perform an action based on the value of the "action" request parameter:

- action=signal: signals an execution path. Additional request parameters: executionPath (required) and transitionName (not required). Returns an XML representation of the execution path.
- action=suspend: suspends a process, no additional parameters needed.
- action=resume: resumes a process, no additional parameters needed.

DELETE: deletes the process instance.

/workflow/processDefinition

GET: retrieves an XML representation of the list of all process definitions, in all their versions. Add a request parameter latestOnly=true to get only the latest version of each process definition.

POST: deploys a new process definition. The payload should be a multipart request containing an item named "processdefinition" which contains the process definition, either in XML form or as a process archive (zip file). The content type of the item should be set accordingly, thus to text/xml or application/zip.

Reloading the sample workflows can be done by using POST with the request parameter action= loadSamples (no body data).

/worfklow/processDefinition/<id>

GET: retrieves an XML representation of this process definition

DELETE: permanently deletes the process definition. Be careful: this also deletes all process instances associated with this definition, without any further warning.

/workflow/processDefinition/<id>/initialVariables

GET: retrieves the (calculated) initial variable values for the start-state task.

/workflow/processDefinitionByName/<name>

GET: retrieves an XML representation of this process definition.

/workflow/pool

GET: retrieves the XML representation of the list of either:

- all pools
- the pools to which a certain user belongs, specified using the request parameter limitToUser=<userId>

POST: creates a new pool, the payload should be an XML document with wf:pool as root element (see pool.xsd)

/workflow/pool/<id>

GET: retrieves an XML representation of the pool

POST: updates the pool by submitting an updated variant of the XML representation retrieved using GET.

DELETE: deletes the pool

/workflow/poolByName/<name>

GET: retrieves an XML representation of the pool.

/workflow/pool/<id>/membership

GET: retrieves the list of users that are member of this pool (as a wf:users element, see pool.xsd).

POST: updates the membership of the pool, according to the value of the "action" request parameter:

- add or remove: adds or removes users to/from the pool. The users should be specified in the XML payload as a wf:users element (see pool.xsd)
- clear: removes all users from the pool

/workflow/query/process

POST: performs a query for processes, the query is described in the XML payload. See Workflow query system (page 288) for more details.

/workflow/query/task

POST: performs a query for tasks, the query is described in the XML payload. See Workflow query system (page 288) for more details.

/workflow/query/timer

POST: performs a query for timers, the query is described in the XML payload. See Workflow query system (page 288) for more details.

/workflow/processInstanceCounts

GET: retrieves the number of instances that exist for each version of each process definition, returned in the form of some XML.

/workflow/timer/*

GET: retrieves an XML representation of the timer.

Notes

1. http://www.jboss.com/products/jbpm
2. http://www.jboss.com/products/jbpm
3. http://www.google.com/search?q=workflow
4. http://www.jboss.com/products/jbpm/docs
5. http://svn.cocoondev.org/viewsvn/trunk/daisy/services/workflow/server-impl/src/processes/
6. javadoc:org.outerj.daisy.repository.Repository
7. javadoc:org.outerj.daisy.workflow.WfActorKey
8. javadoc:org.outerj.daisy.workflow.WfVersionKey
9. javadoc:org.outerj.daisy.workflow.WfUserKey
10. javadoc:org.outerj.daisy.workflow.WfVersionKey
11. javadoc:org.outerj.daisy.repository.Repository
12. javadoc:org.outerj.daisy.repository.Repository
13. javadoc:org.outerj.daisy.repository.RepositoryManager
14. http://freemarker.org/
15. javadoc:root
16. javadoc:org.outerj.daisy.workflow.WorkflowManager

9 Administration

9.1 Starting and stopping Daisy

This document contains a summary of how to start and stop Daisy after the initial installation (page 24).

9.1.1 Starting Daisy

The order in which the processes are started is important, and should be the order in which they are mentioned below.

The command snippets below assume **the JAVA_HOME and DAISY_HOME environment variables** are set. If they are not set globally, set them each time before executing the commands.

9.1.1.1 Start MySQL

Start MySQL if necessary. Usually MySQL is started together with the operating system (in Windows as a service, in Linux using the /etc/init.d/mysql script), so you don't need to look after this.

9.1.1.2 Start the Daisy Repository Server

9.1.1.2.1 Windows

```
cd %DAISY_HOME%\repository-server\bin
daisy-repository-server <daisydata-dir>
```

9.1.1.2.2 Linux/Unix/MacOSX

```
cd $DAISY_HOME/repository-server/bin
./daisy-repository-server <daisydata-dir>
```

9.1.1.3 Start the Daisy Wiki

9.1.1.3.1 Windows

```
cd %DAISY_HOME%\daisywiki\bin
daisy-wiki <wikidata-dir>
```

9.1.1.3.2 Linux/Unix/MacOSX

```
cd $DAISY_HOME/daisywiki/bin
./daisy-wiki <wikidata-dir>
```

9.1.2 Stopping Daisy

The processes should by preference be stopped in reverse order of how they were started. Thus: first the Daisy Wiki, and then the Daisy Repository Server, then possibly MySQL.

There are no special mechanisms (scripts) to stop the Daisy Wiki and the Daisy Repository Server. If they are running in a console, just press Ctrl+C. Otherwise, on Linux/Unix/MacOSX you can kill the process (not a kill -9 but a normal kill, this will nicely end them).

9.1.3 The better way to do all this

A nice way to start and stop the various processes is by using the wrapper from tanukisoft[1]. See running Daisy as a service (page 301).

9.2 Running Daisy as a service

9.2.1 Concept and general instructions

If you want to run Daisy as part of your operational environment, you might prefer to run the different Daisy components as *services*, which you can easily start, stop and restart, and which ideally run under a specific user (perhaps with restricted privileges). We recommend using "wrapper[2]" for this, a fine open source project from Tanuki Software. If you use wrapper and are happy with it, consider making a donation[3] to its developers.

Tanuki wrapper can be used *both* on Unix and Windows platforms. As of Daisy 2.0, the wrapper scripts are integrated into the daisy binary distribution. The following sections explain the implementation details and how to make use of these scripts.

Installing the scripts

(starting from Daisy 2.0.1)

Run the following command

```
<daisy.home>/install/daisy-service-install -r <repo-data-dir> -w <wiki-data-dir>
```

You can choose to give only the -r or -w option, as desired. To see all available options, specify the `-h` (help) option.

The service scripts will then be installed in the service subdirectory of the repository and wiki data directories.

Implementation details

What this command does is copying some template flies from the `DAISY_HOME/wrapper` directory, and adjusting some paths in the conf files.

Using the wrapper scripts

Detailed instructions on the usage of the wrapper scripts have been created for the following platforms:

- Unix (page 303)
- Windows (page 304)

Using a custom jetty-daisywiki.xml

If you have a custom jetty-daisywiki.xml in your wiki data directory, you will need to edit the following file:

```
<wiki data dir>/service/daisy-wiki-service.conf
```

Instructions are in the file itself.

Relocating the data directories or changing the location of daisy home

If you would like to relocate either your daisy data directory or your daisy wiki data directory, or if you want to move your daisy home directory, you have to either manually adapt the settings inside the wrapper configuration files, or run `daisy-service-install` again. What follows are instructions for manual adjustment.

The repository startup script is configured via the file `<DAISY_DATADIR>\service\daisy-repository-server-service.conf`, at the very beginning of that file, you will find the following section:

```
# Environment variables
set.default.DAISY_HOME=/path/to/your/daisy/home
set.DAISY_DATADIR=/path/to/your/daisy/datadir
set.WRAPPER_HOME=%DAISY_HOME%/wrapper
```

Simply replace the path for `DAISY_HOME` and/or `DAISY_DATADIR`, and you are done.

In case of the daisy wiki startup script, configuration takes place via `<DAISYWIKI_DATADIR>\service\daisy-wiki-service.conf`. At the very beginning of that file, you will find the following section:

```
# Environment variables
set.default.DAISY_HOME=/path/to/your/daisy/home
set.default.DAISYWIKI_DATADIR=/path/to/your/daisywiki/datadir
set.JETTY_HOME=%DAISY_HOME%/daisywiki/jetty
set.WRAPPER_HOME=%DAISY_HOME%/wrapper
```

Simply replace the path for `DAISY_HOME` and/or `DAISYWIKI_DATADIR`, and you will be able to use the script again.

Usage of the wrapper scripts on other platforms

For MacOS and Solaris, precompiled binaries of the wrapper scripts binaries and libraries are shipped together with the daisy binary distribution, so that the wrapper scripts should run out of the box on these platforms .

In order to use the wrapper scripts on AIX or HP-UX, you have to perform the following steps:

- Download the wrapper binaries for your platform from the wrapper's download page[4] at sourceforge (e.g. `wrapper-aix-ppc-32-3.2.x.tar.gz`).
- Copy the file `wrapper` inside the `bin`-directory of your downloaded wrapper archive to the directory $DAISY_HOME/wrapper/bin of your daisy installation.
- Copy the wrapper library file (e.g. `libwrapper.a`) which can be found inside the `lib`-directory of your downloaded wrapper archive to the directory $DAISY_HOME/wrapper/lib of your daisy installation.

That's it already. You now hopefully will be able to start up the services by using the startup files in the service directory of your daisy data directory or your daisywiki data directory, respectively. In case the startup fails, you might need to adapt the variable `WRAPPER_CMD` inside the startup files.

Hope these instructions helps as a quick primer on how to use wrapper with Daisy. Wrapper's documentation is quite good, so please review it before asking questions.

Wrapper also monitors the JVM processes by *"pinging"* them regularly, and will even restart the processes if a VM gets stuck - which, under normal circumstances, shouldn't happen.

9.2.2 Running services on Unix

9.2.2.1 Manually starting, stopping and restarting the service scripts

Manually starting, stopping or restarting the services is done via wrapper scripts, provided in the `services` directory of either your repository data directory or your wiki data directory:

In case you have no services directory, see the service installation instructions (page 301).

9.2.2.1.1 Daisy Repository

To manually startup, stop or restart the daisy repository, simply run the wrapper script `daisy-repository-server-service` inside the directory `service` below your daisy data directory, using the correct parameter (start | stop | restart):

```
$cd /path/to/your/daisy/datadir/service
$ ./daisy-repository-server-service
Usage: ./daisy-repository-server-service { console | start | stop | restart | dump }
$ ./daisy-repository-server-service start
Starting Daisy CMS Repository Server instance...
```

9.2.2.1.2 Daisy Wiki

To manually startup, stop or restart the daisy wiki, simply run the wrapper script `daisy-wiki-service` inside the directory `service` below your daisy wikidata directory, using the correct parameter (start | stop | restart):

```
$cd /path/to/your/daisy/datadir/service
$ ./daisy-wiki-service
Usage: ./daisy-wiki-service { console | start | stop | restart | dump }
$ ./daisy-wiki-service start
Starting CMS Wiki Web Application instance...
```

9.2.2.2 Installation as Unix service

TODO: the following doesn't handle the required sleep between starting the repository and the wiki.

In order to install the daisy repository and/or the daisy wiki as service, simply create links from inside the directory /etc/init.d, points to the wrapper scripts `daisy-repository-server-service` and `daisy-wiki-service`, provided in the `services` directory of either your daisy data directory or your daisy wikidata directory. Afterwards, place links to the newly created link in the according runlevel directories.

Alternatively, you may use the command `chkconfig` in order to implement the unix services. For further information, consult the man page.

9.2.2.3 Testing and debugging the service scripts

In case of problems or startup failures of the services, you may have a look at the log files `daisy-repository-server-service.log` and `daisy-wiki-service.log` which can be found in the `logs` directory of either your daisy data directory or your daisy wiki data directory.
Also, you may start the wrapper scripts interactively, so that the output of the process is redirected to the console. This is simply done by using the parameter console when manually invoking the wrapper scripts `daisy-repository-server-service` and `daisy-wiki-service`, provided in the `services` directory of either your daisy data directory or your daisy wikidata directory.

9.2.3 Running services on Windows

9.2.3.1 Installing the windows services

Installation of the services is done via a batch script, provided in the `services` directory of either your daisy data directory or your daisy wiki data directory:

In case you have no services directory, see the service installation instructions (page 301).

9.2.3.1.1 Daisy Repository

To install the daisy repository as windows service, simply run the batch file `install-daisy-repository-server-service.bat` inside the directory `service` below your daisy data directory:

```
C:\>cd C:\Path\to\your\daisy\datadir\service
C:\Path\to\your\daisy\datadir\service>install-daisy-repository-server-service.bat
wrapper  | Daisy CMS Repository Server installed.
```

Alternatively, you can point your Windows-Explorer to the directory `service` below your daisy data directory and double click on the file `install-daisy-repository-server-service.bat` in order to launch the installation.

9.2.3.1.2 Daisy Wiki

To install the daisy wiki as windows service, run the batch file `install-daisy-wiki-service.bat` inside the directory `service` below your daisywiki data directory:

```
C:\>cd C:\Path\to\your\daisy\wikidatadir\service
C:\Path\to\your\daisy\wikidatadir\service>install-daisy-wiki-service.bat
wrapper  | Daisy CMS Wiki Web Application installed.
```

Alternatively, the batch file can be started via double click inside the Windows Explorer, like explained above.

Please note that after each change of the wrapper configuration files, you have to reinstall the wrapper services in order to take the changes effect.

9.2.3.2 Starting, stopping and restarting the services

Starting, stopping or restarting of the services is done via batch scripts, provided in the `services` directory of either your daisy data directory or your daisy wikidata directory:

9.2.3.2.1 Daisy Repository

To startup the daisy repository as windows service, simply run the batch file `start-daisy-repository-server-service.bat` inside the directory `service` below your daisy data directory:

```
C:\>cd C:\Path\to\your\daisy\datadir\service
C:\Path\to\your\daisy\datadir\service>start-daisy-repository-server-service.bat
Daisy CMS Repository Server will be launched...
Daisy CMS Repository Server was launched successfully.
```

Alternatively, you can point your Windows-Explorer to the directory `service` below your daisy data directory and double click on the file `install-daisy-repository-server-service.bat` in order to launch the installation.

For stopping or restarting the daisy repository service, simply use the files `start-daisy-repository-server-service.bat` or `start-daisy-repository-server-service.bat`, respectively.

If the name of your MySQL service is different from `MySQL`, you will end up with system error 1075, complaining about the fact that a dependent service does not exist. In that case simply adjust the value of the property `wrapper.ntservice.dependency.1` inside the config file `daisy-repository-server-service.conf` to the name of your MySQL service (e.g. `MySQL5`).

9.2.3.2.2 Daisy Wiki

To startup the daisy wiki as windows service, simply run the batch file `start-daisy-wiki-service.bat` inside the directory `service` below your daisy wikidata directory:

```
C:\>cd C:\Path\to\your\daisy\wikidatadir\service
C:\Path\to\your\daisy\wikidatadir\service>start-daisy-wiki-service.bat
Daisy CMS Wiki Web Application will be launched...
Daisy CMS Wiki Web Application was launched successfully.
```

Alternatively, the batch file can be started via double click inside the Windows Explorer, like explained above.

For stopping or restarting the daisy wiki service, simply use the files `start-daisy-wiki-service.bat` or `start-daisy-wiki-service.bat`, respectively.

9.2.3.3 Uninstalling the windows services

Deinstallation of the services is done via a batch script, provided in the `services` directory of either your daisy data directory or your daisy wiki data directory:

9.2.3.3.1 Daisy Repository

To uninstall the daisy repository as windows service, simply run the batch file `uninstall-daisy-repository-server-service.bat` inside the directory `service` below your daisy data directory:

```
C:\>cd C:\Path\to\your\daisy\datadir\service
C:\Path\to\your\daisy\datadir\service>uninstall-daisy-repository-server-service.bat
wrapper  | Daisy CMS Repository Server removed.
```

Alternatively, you can point your Windows-Explorer to the directory `service` below your daisy data directory and double click on the file `uninstall-daisy-repository-server.service.bat` in order to launch the deinstallation.

You may also use the Windows Service console in order to start, stop or restart the daisy repository service or change the characteristics of that service. In order to invoke that console, goto **Start|Run** and enter “services.msc” or invoke the console from the system panel.

9.2.3.3.2 Daisy Wiki

To uninstall the daisy wiki as windows service, run the batch file `uninstall-daisy-wiki-service.bat` inside the directory `service` below your daisywiki data directory:

```
C:\>cd C:\Path\to\your\daisy\wikidatadir\service
C:\Path\to\your\daisy\wikidatadir\service>uninstall-daisy-wiki-service.bat
wrapper  | Daisy CMS Wiki Web Application removed.
```

Alternatively, the batch file can be started via double click inside the Windows Explorer, like explained above.

You may also use the Windows Service console in order to start, stop or restart the wiki service. Invokation of that console is described above.

9.2.3.4 Testing and debugging the service scripts

In case of problems or startup failures of the services, you may have a look at the log files `daisy-repository-server-service.log` and `daisy-wiki-service.log` which can be found in the `logs` directory of either your daisy data directory or your daisy wiki data directory.
Also, you may start the wrapper scripts interactively, so that the output of the process is redirected to the console. This is explained below:

9.2.3.4.1 Daisy Repository

In order to debug the daisy repository, simply run the batch file `daisy-repository-server-service.bat` inside the directory `service` below your daisy data directory:

```
C:\>cd C:\Path\to\your\daisy\datadir\service
C:\Path\to\your\daisy\datadir\service>daisy-repository-server-service.bat
wrapper  | --> Wrapper staretd as console
wrapper  | --> Launching a JVM...
jvm 1    | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1    |   Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.
... console output will follow here
```

Make sure that the %JAVA_HOME% variable is set as system variable and not as user variable. Otherwise you may end up with an system error 1067 while starting up the repository via the service script.

9.2.3.4.2 Daisy Wiki

In order to debug the daisy wiki, simply run the batch file `daisy-wiki-service.bat` inside the directory `service` below your daisy data directory:

```
C:\>cd C:\Path\to\your\daisy\datadir\service
C:\Path\to\your\daisy\datadir\service>daisy-wiki-service.bat
wrapper  | --> Wrapper staretd as console
wrapper  | --> Launching a JVM...
jvm 1    | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1    |   Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.
... console output will follow here
```

9.3 Deploying on Tomcat

The Daisy webapp can be freely moved out of the `DAISY_HOME/daisywiki` directory and deployed on another servlet container. It has *no* dependencies on the other stuff from `DAISY_HOME`, but it does need access to the daisywiki data directory (not the repository data directory). It does *not* need access to the relational database. See also the diagram in the installation instructions (page 24).

Here is how you can deploy Daisy on Tomcat (this was tried with Tomcat 5.0.28, 5.5.17 and 6.0.10):

First copy Daisy's webapp (directory `DAISY_HOME/daisywiki/webapp`) to `TOMCAT_HOME/webapps/cocoon` (thus the `webapp` directory gets renamed to `cocoon`).

Then make sure recent versions of xerces and xalan are in the endorsed library path of Tomcat, you can do this like this (in the Tomcat directory):

```
$ ls common/endorsed/
xercesImpl.jar  xml-apis.jar
$ rm common/endorsed/*
$ cp webapps/cocoon/WEB-INF/lib/xercesImpl-<version>.jar common/endorsed/
$ cp webapps/cocoon/WEB-INF/lib/xml-apis.jar common/endorsed/
$ cp webapps/cocoon/WEB-INF/lib/xalan-<version>.jar common/endorsed/
```

To avoid encoding problems, edit the file `TOMCAT_HOME/conf/server.xml`, search for the element `<Connector port="8080"` ... and add to it the following two attributes:

```
URIEncoding="UTF-8" useBodyEncodingForURI="true"
```

Also, make sure your file encoding is set to UTF-8 (e.g. via the CATALINA_OPTS environment variable):

```
export CATALINA_OPTS=-Dfile.encoding=UTF-8
```

To inform the webapp of the location of the wikidata directory, a system property `daisywiki.data` should be supplied. This can be done as follows:

```
Windows:
set CATALINA_OPTS=-Ddaisywiki.data=<path to wikidata directory>

Linux:
export CATALINA_OPTS=-Ddaisywiki.data=<path to wikidata directory>
```

However, this does not work if you want to deploy the Daisy Wiki webapp multiple times inside the same servlet container, using different wikidata directories. In that case, you might either specify the `daisyswiki.data` parameter inside your WEB-INF/web.xml (page 317), or inside the context file CATALINA_HOME/conf/Catalina/your.domain.tld/ROOT.xml. If you prefer to stick to the latter method, which has the advantage that it will survive an update of your daisy installation, your context file should look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context reloadable="true">
  <!-- specify wiki data directoty -->
  <Parameter name="daisywiki.data" value="/path/to/your/daisy/wiki/data"
override="false" />
  ...
</Context>
```

See Specifying the wikidata directory location (page 317) for more details.

Now start Tomcat, and surf to http://localhost:8080/cocoon/daisy/. That's all there is to it. Note that Daisy requires an expanded webapp, not a war archive.

9.4 Changing location (port or machine) of the different processes

This section details the changes to be done to run the different servers needed for Daisy on different machines or let them listen to different ports.

The machine on which you want to run the Daisy Wiki only needs the `DAISY_HOME/daisywiki` subdirectory. The machine on which you want to run the repository server needs at least `DAISY_HOME/lib` and `DAISY_HOME/repository-server`, in addition to your daisy data directory. The install utility programs need `DAISY_HOME/install` and `DAISY_HOME/lib`.

After editing any of the configuration files, you need to restart the application to which the configuration file(s) belongs.

9.4.1 Running MySQL at a different location

Both ActiveMQ and the Daisy Repository Server talk to MySQL, so the configuration of both needs to be adjusted. Both could use a different MySQL instance.

For ActiveMQ, edit the following file:

```
<daisydata directory>/conf/activemq-conf.xml
```

Look for the <property name="url" value="..."/> element.

For the Daisy Repository Server, edit the following file:

```
<daisydata directory>/conf/myconfig.xml
```

Look for the override of the datasource configuration:

```
<target path="/daisy/datasource/datasource">
```

and adjust there the content of the <url> element.

9.4.2 Running ActiveMQ on a different port

The port to which ActiveMQ listens (by default 61616) can be changed in the ActiveMQ configuration:

```
<daisydata directory>/conf/activemq-conf.xml
```

Since ActiveMQ is embedded in the repository server, the repository server itself does not communicate via a TCP connection and thus needs no configuration change.

For the Daisy Wiki: edit the following file:

```
<wikidata directory>/daisy.xconf
```

Search for the following element:

```
<property name="java.naming.provider.url" value="tcp://localhost:61616"/>
```

and adjust.

9.4.3 Running the Daisy Repository Server at a different location

If you want to change the default HTTP port (9263) of the Daisy Repository Server, edit the following file:

```
<daisydata directory>/conf/myconfig.xml
```

Look for the following element and adjust:

```
<port>9263</port>
```

The Daisy Wiki talks to Daisy Repository Server, so if you change the port or run the repository server on a different machine, edit the following file:

```
<wikidata directory>/daisy.xconf
```

Look for the following element and adjust:

```
<repository-server-base-url>http://localhost:9263</repository-server-base-url>
```

9.4.4 Changing the JMX console port

Edit the following file:

```
<daisydata directory>/conf/myconfig.xml
```

Look for the following element and adjust:

```
<httpAdaptor port="9264"/>
```

9.4.5 Changing the Daisy Wiki (Jetty) port

Copy the default jetty configuration to the wiki data directory

```
cp DAISY_HOME/daisywiki/conf/jetty-daisywiki.xml WIKI_DATA/jetty_daisywiki.xml
or
copy DAISY_HOME/daisywiki/conf/jetty-daisywiki.xml WIKI_DATA/jetty_daisywiki.xml
```

Edit your newly copied file and look for the following line and adjust the '8888' value:

```
<Set name="Port"><SystemProperty name="jetty.port" default="8888"/></Set>
```

If you are using the tanuki wrapper scripts you must point jetty in the direction of your new settings. Comment the default settings and uncomment the custom settings

```
# Default Daisy Jetty configuration
# set.DAISY_JETTY_CONF=%DAISY_HOME%/daisywiki/conf/jetty-daisywiki.xml
# Override Jetty configuration (uncomment the line below to override default
configuration)
set.DAISY_JETTY_CONF=%DAISYWIKI_DATADIR%/jetty-daisywiki.xml
```

9.5 Repository Administration

Administrative actions for the Daisy Repository Server include:

- managing users
- managing document collections
- managing document, part and field types (ie, the repository schema)
- managing access control

All these operations should be performed through the Daisy Repository Server, never directly on the SQL database. This is on the one hand because the repository server caches a good deal of these things, and wouldn't otherwise be aware of these changes, and also because it is too error-prone to directly change them on the database. Furthermore, JMS events of these operations are generated, which would otherwise not be engendered.

All the management operations can be performed through the Java API (page 112) or through the HTTP+XML interface (page 119). The Daisy Wiki front end contains a set of *administration screens* (based on these APIs) that allow you to perform all these operations through a web browser.

To perform these operations you need to be logged in with a user having the active role of "Administrator" (a predefined role). See also User Management (page 82).

9.6 Emailer Notes

When Daisy sends emails (this is e.g. done by the email notifier (page 88), the user self-registration component, the workflow task notifiactions), the emails are first queued in a database table. The purpose of this is to decouple the component that sends email from the actual sending of the emails.

The database table used for this is called email_queue, and has the following structure:

```
mysql> show columns from email_queue;
+---------------+--------------+------+-----+---------------------+----------------+
| Field         | Type         | Null | Key | Default             | Extra          |
+---------------+--------------+------+-----+---------------------+----------------+
| from_address  | varchar(255) | YES  |     | NULL                |                |
| to_address    | varchar(255) |      |     |                     |                |
| subject       | varchar(255) |      |     |                     |                |
| message       | mediumtext   |      |     |                     |                |
| retry_count   | tinyint(4)   |      |     | 0                   |                |
| last_try_time | datetime     | YES  |     | NULL                |                |
| created       | datetime     |      |     | 0000-00-00 00:00:00 |                |
| id            | bigint(20)   |      | PRI | NULL                | auto_increment |
| error         | varchar(255) |      |     |                     |                |
+---------------+--------------+------+-----+---------------------+----------------+
```

The emailer component checks regularly (by default: every 20 seconds) if new records have been inserted. It then sends the corresponding email for each of these new records, if a mail is successfully sent (i.e., handed over to the smtp server), the record gets deleted from the database table. If it fails, the exception message is put in the error column, the retry_count column is augmented, and the last_try_time column gets the current time.

The emailer component will try up to 3 times to send a mail, each time waiting 5 hours in between (these values are configurable), before giving up. The record for that email then keeps sitting in the database table for a certain time (by default 7 days), after which it is removed without further notice.

If mails couldn't get out because of a configuration error, or because the smtp server was down, and you want to force the mails to be resend, you can do this by resetting the retry_count and last_try_time columns:

```
update email_queue set retry_count = 0, last_try_time = null;
```

9.7 Log files

9.7.1 Repository server

The log files of the repository server are by default created in the following directory:

```
<daisydata directory>/logs
```

The log files rol over daily. You will probably want to regularly delete old log files, for example using a cron job. The logging configuration can be adjusted in the following file:

```
<daisydata directory>/conf/repository-log4j.properties
```

There are two log files:

- **daisy-request-errors-<date>.log**: in this file all exceptions are logged that occur during the processing of external requests (thus requests over the HTTP interface).
- **daisy-<date>.log**: this file contains all other log output, such as generated during startup of the server or by background threads.

9.7.2 Daisy Wiki

The log files of the Daisy Wiki are created in the following directory:

```
<wikidata directory>/logs
```

9.8 Running Apache and Daisy

Why running Apache in front of Daisy?

Daisy ships with Jetty[5], which is a nice, compact, and fast servlet container. The Daisy Wiki application can be deployed in Tomcat as well (page 307). However, many people (ourselves included) prefer to run Apache in front of the Daisy servlet container, as this provides the ability to listen to the usual port 80 number, without requiring to run Jetty or Tomcat as root processes. So we need some way to hook up Apache and Daisy, which is described in the document underneath.

Required Apache modules

First of all, we need to install a recent version of Apache (v2), with the **mod_proxy**[6] and **mod_rewrite**[7] modules activated. Most likely, there will be a binary build of Apache for your platform available - if not, the process of building Apache and both modules is explained on the Apache httpd website[8]. Apparently, the mod_proxy implementation inside Apache v1 isn't considered very robust, so be sure to install Apache v2.

Configuring Apache

In this example, I'm running my website as a Virtual Host in my Apache config, but these instructions should also apply when using Apache to serve only one website: only the location of the directives will be slightly different. Here's what our server configuration looks like:

```
NameVirtualHost *:80
<VirtualHost *:80>
        ServerName cocoondev.org
        ServerAdmin webmaster.at.cocoondev.org

        DocumentRoot /home/daisy_cd

        RewriteEngine on
        RewriteRule "^/WEB-INF/?(.*)" "$0" [L,F,NC]

        RewriteRule "^/$" http://localhost:8080/main/ [P]
        RewriteRule "^/(.*)" "http://localhost:8080/$1" [P]

        ProxyRequests Off
        ProxyPassReverse / http://localhost:8080/
        ProxyPreserveHost On

        ErrorLog /var/log/apache2/error.log
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined

        ServerSignature On

</VirtualHost>
```

DocumentRoot

I picked the home directory of the user running Daisy as the Apache DocumentRoot. As all requests will be proxy-forwarded anyhow, this setting isn't particularly relevant.

Securing the /WEB-INF/ directory

Though Jetty doesn't service requests starting with /WEB-INF/ by default, we make sure such requests are blocked on the Apache level as well:

```
        RewriteRule "^/WEB-INF/?(.*)" "$0" [L,F,NC]
```

Here, the [F] flag forbids accessing the matched resource. [NC] makes the pattern case-insensitive.

Getting rid of the /daisy/ prefix

Typically, you don't want to see the `/daisy/` prefix appear in a Daisy-based website. That requires you to edit Daisy's Cocoon sitemap (we might come up with a different solution for this in the future). Edit `$DAISY_HOME/daisywiki/webapp/sitemap.xmap`:

Around line 578, near the `<!-- welcome page -->` comment, replace:

```
<map:match pattern="">
```

with

```
<map:mount check-reload="yes" src="daisy/" uri-prefix=""/>
```

Then remove the following lines until you reach

```
<map:handle-errors>
```

which should be around line 718.

Proxying Jetty

mod_rewrite can also pass the handling of requests to the mod_proxy module (using the [P] flag), and we make use of this in the statement which makes sure all requests are simply forwarded from Apache to Jetty, where they will be picked up by Cocoon for further processing:

```
        RewriteRule "^/(.*)" "http://localhost:8080/$1" [P]

        ProxyRequests Off
        ProxyPassReverse / http://localhost:8080/
        ProxyPreserveHost On
```

By turning ProxyRequests Off, we also make sure our web server can't be used as an open proxy for people who want to surf anonymously. Make sure you don't forget to add the ProxyPreserveHost directive, which is especially needed in Virtual Hosts serving context.

Final touches

Normally, Daisy's main access point is the page listing the different sites available (depending on login and role) from that specific Daisy instance. If we want to skip this page, and forward visitors to one specific site, consider adding this rule to the server config:

```
        RewriteRule "^/$" http://localhost:8080/main/ [P]
```

This way, if someone hits the root of the website, he will be automatically forwarded to the homepage of the 'main' Daisy site (http://host/main/). The other sites are still available, by simply accessing http://host/otherdaisysite/. Also, the administration screens are accessible using http://host/admin/.

Acknowledgments

Kudos to Pier for describing[9] this setup technique on the Cocoon Wiki. This setup is in use at a very high profile / high volume news website, so it should work for Daisy as well.

9.9 Configuring upload limits

Daisy has no arbitrary limits on the size of the document parts, the eventual limits are the available disk space, file system limitations, and the java.lang.Long.MAX_VALUE value.

However, the Daisy Wiki and the Daisy Repository Server have configurable limits of how much data one is allowed to upload. For the Daisy Wiki, this is by default 10 MB, for the Repository Server, this is by default 400 MB.

The Daisy Wiki limitation can be configured in the file `daisywiki/webapp/WEB-INF/web.xml`, search for `upload-max-size`.

The Repository Server limit is configurable via the `<daisydata-dir>/conf/myconfig.xml` file, by including a section like this:

```
<target path="/daisy/repository/httpconnector">
  <configuration>
      <upload>
        <!-- threshold beyond which files are directly written to disk (in bytes) -->
        <threshold>50000</threshold>
        <!-- Maximum upload size (in bytes), -1 for no maximum -->
        <maxsize>400000000</maxsize>
        <!-- Location for temporary upload files, default java.io.tmpdir -->
        <!-- tempdir></tempdir -->
      </upload>
  </configuration>
</target>
```

Note that when uploading a file via the Daisy Wiki, you might need up to three times the size of the file available on disk: first the file is uploaded to the Daisy Wiki which uses a temporary file to store it, then Daisy Wiki uploads it to the repository server, which again uses a temporary file to store it, and finally the repository server stores the file in its "blobstore".

9.10 Include Permissions

Daisy allows to do inclusions in documents. This can be either other Daisy documents (through the "daisy:" syntax) or when published through the Daisy Wiki, any URL resolvable by Cocoon (the framework on which the Daisy Wiki is implemented). However, allowing document authors to include any arbitrary URL might introduce security issues. For example, one could retrieve content from the file system using a "file:" URL. Also, the "http:" URLs are resolved behind the firewall (if any) which might be a problem too.

Therefore, by default, all external inclusions are disallowed. You can give access for certain schemes or URLs through a configuration file. This file is located at:

```
<wikidata directory>/external-include-rules.xml
```

This file has a <rules> root element containing <rule> children:

```
<?xml version="1.0"?>
<rules>
  <rule ... />
  <rule ... />
</rules>
```

Each rule element can have the following attributes:

- `scheme`: http, file, cocoon, etc.
- `value`: the value which should be checked, is one of: `uri`, `host`, `ip` (host and ip do not make sense for all schemes)
- `matchType`: how the value should be matched, one of:
  - string
  - stringIgnoreCase
  - wildcard
  - wildcardIgnoreCase
  - regexp
  - regexpIgnoreCase
  - subdir (see further on for more details)

- `matchValue`: the value (or regexp or wildcard expr) to be matched against
- `portFrom` (optional, only for HTTP)
- `portTo` (optional, only for HTTP)
- `permission`: grant or deny

There are two possibilities:

- either the <rule> element has only the scheme and permission attributes, to globally allow a scheme
- or it has all attributes (except for the optional portFrom and portTo)

Evaluation of the rules:

- start situation: all inclusions are denied.
- all rules are processed, top to bottom, later results overwriting earlier results.

Example:

```
<?xml version="1.0"?>
<rules>
  <rule scheme="http" value="uri" matchType="stringIgnoreCase"
        matchValue="http://www.somehost.com/somefile.xml" permission="grant"/>
  <rule scheme="http" value="ip" matchType="wildcard" matchValue="67.19.2.*"
        portFrom="80" portTo="80" permission="grant"/>
  <rule scheme="http" value="host" matchType="stringIgnoreCase"
        matchValue="blogs.cocoondev.org" permission="deny"/>
  <rule scheme="cocoon" value="uri" matchType="wildcardIgnoreCase"
        matchValue="cocoon://daisy/ext/**" permission="grant"/>
  <rule scheme="file" value="uri" matchType="subdir"
        matchValue="/home/bruno/tmp" permission="grant"/>
</rules>
```

Some more notes on the matchtypes:

- `wildcard` uses the cocoon wildcard matcher. This means one * matches any character except for slash ("/"), while ** matches any character
- `regexp` uses the default Java regexp engine. You will usually want to start them with ^ and end them with $.
- `subdir` is a special case, which should only be used with the file scheme. It constructs a java.io.File object for the value and the matchValue, takes their canonical path (which is absolute and unique), and check whether the first starts with the second.

Further notes:

- changes to the external-include-rules.xml file are immediately detected, thus do not need a restart of the Daisy Wiki.

9.11 Going live

This section contains items that might be worthwhile to check when putting a Daisy instance into production.

9.11.1 Change the "testuser" account

When configuring the repository server using the "daisy-repository-init" script, you are prompted for an initial bootstrap user account to be able to connect to the repository. By default the username "testuser", password "testuser" is proposed, though the installation script suggest to create a first real user here. Experience shows many people leave it to the testuser/testuser defaults.

Therefore, before going live, be sure to protect this account by changing the password to something else, or by unchecking the "registration confirmed" checkbox.

9.11.2 More?

Suggestions welcome.

9.12 Large repositories

If you have more than 10 000 documents in your Daisy repository, it is recommended to enlarge the document cache size in order to keep good performance. Ideally, the cache should be as large as the number of documents (or the set of frequently accessed documents thereof) .

To configure the document cache, add the following to the myconfig.xml (page 0):

```
<target path="/daisy/repository/documentcache">
  <configuration>
      <documentCacheMaxSize>15000</documentCacheMaxSize>
      <availableVariantsCacheMaxSize>15000</availableVariantsCacheMaxSize>
  </configuration>
</target>
```

If you enlarge the cache size, you should also enlarge the maximum amount of memory available to the repository server JVM.

9.13 Specifying the wikidata directory location

In a default Daisy setup, the location of the wikidata directory is communicated to the Daisy Wiki by means of a Java system property (this is a property passed to the Java executable using a '-D' argument). The name of the property is `daisywiki.data`.

This is convenient for the default setup as it allows to specify the wikidata directory location as a command line argument. However, this doesn't allow to deploy the Daisy Wiki webapp multiple times in the same servlet container, using different wikidata directories. Or sometimes people might not be able to control the system properties passed to the servlet container.

For such cases, it is also possible to specify the wikidata directory location using a Servlet context initialization parameter. Such parameter can be specified in the WEB-INF/web.xml file (part of the webapp, requires to make multiple duplicates of the webapp directory), or outside of the webapp using a servlet-container specific configuration mechanism. The context initialization property is also named `daisywiki.data`.

Here is an example of specifying it in the web.xml:

```
<web-app>

  <context-param>
```

```
    <param-name>daisywiki.data</param-name>
    <param-value>/path/to/wikidata</param-value>
  </context-param>

... rest of the config ...

</web-app>
```

If the daisywiki.data property is specified both as a system property and Servlet context initialization parameter, then the context initialization parameter gets precedence.

9.14 Making backups

9.14.1 Introduction

For backing up Daisy there is an application which will help you create and restore backups. You can find it in

```
<DAISY_HOME>/bin/daisy-backup-tool
```

This tool in its current form can backup :

- daisy-repository
    - repository database
    - blobstore
    - indexstore
    - configuration
- activemq database (ActiveMQ is the JMS system)
- daisy-wiki
    - cocoon.xconf
    - web.xml
    - skins
    - sites
    - you name it (this will be elaborated on further in this document (page 321))

Your backup will contain database dumps of the configured databases and zip files containing the content of directories or files.

9.14.2 Running the backup tool

9.14.2.1 Prerequisites

The following DBMS can be backed up :

- MySQL (binaries used : mysql, mysqldump, mysqladmin)

MySQL 4.0 users will have to manually edit the database dump. First unzip the dump file. Add "SET FOREIGN_KEY_CHECKS=0;" in the beginning of the dump and "SET FOREIGN_KEY_CHECKS=1;" at the end. Rezip the dump file. Then run backup-tool with option **-R** to rehash the backup entries.

- PostgreSQL (binaries used : psql, pg_dump, dropdb)

For successful completion of a backup it is required that the binaries can be found in the environments PATH variable.

PostgreSQL backup and/or restore has not been tested yet. Feedback on this subject is always welcome.

9.14.2.2 Running

Running the tool without any parameters will just display the help which should look something like this:

```
usage: daisy-backup-tool [-d <daisydata-path>] -R <backup-name> | -r
       <backup-name> | -h | -b | -v [-a <entry-configuration-file>]   [-f
       <from-address>] [-l <backup-path>]  [-s <smtp-server>] [-e
       <email-address>][-q]
 -d,--daisy-data-dir <daisydata-path>                 Daisy data directory
 -a,--additional-entries <entry-configuration-file>   Path to
                                                      configuration of additional backup
 entries
 -s,--smtp-server <smtp-server>                       Smtp server
 -q,--quiet-restore                                   Suppress confirmation dialog while
                                                      restoring an existing backup
 -R,--rehash <backup-name>                            Rehash files from an
                                                      existing backup
 -b,--backup                                          Create a new backup
 -e,--emailaddress <email-address>                    Where emails will be
                                                      sent to in case of an exception
 -f,--fromaddress <from-address>                      Sender address to
                                                      use for exception emails
 -h,--help                                            Show this message
 -l,--backuplocation <backup-path>                    Location where
                                                      backups are stored
 -r,--restore <backup-name>                           Restore an existing
                                                      backup
 -v,--version                                         Print version info
```

9.14.2.3 Compulsory options

9.14.2.3.1 -b, -r <backup-name> or -R <backup-name>

- The **-b** option is used to create a backup.
- Specifying the **-r <backup-name>** option will allow you to restore a backup. The name of the backup can be found in the backup location directory. You will be looking for a directory name of the form <DATE>_<SEQUENCE> or YYYYMMDD_### (e.g. 20051124_001), the directory name is thus the same name as the backup.
- **-R <backup-name>** rehashes the backup entries. This is used when a user wishes to modify a file before restoring it. Some versions of mysql for example do not switch off foreign key

checks prior to restoring the database. The user should then switch of the checks at the beginning of the database dump and turn them on again at the end. This will modify the file which in turn will cause the tool to complain at restore time saying that a file has been tampered with. Rehashing will then recreate hashes for the modified files allowing the tool to restore without complaining.

Minimally the Daisy repository will be backed up or restored. By specifying different options you are able to backup or restore also files of the Daisy Wiki.

9.14.2.3.2 -l <backup-path>

Specifies where the backups will be created in, restored from, or rehashed.

9.14.2.3.3 -d <daisydata-path> (when using -b or -r)

This option is mandatory when using **-b** or **-r**. The daisydata-path is where your blobstore, indexstore, configuration and logs are stored.

The -d option is required for -r (restore) starting from Daisy 1.5-M2. This is because from that release on, the myconfig.xml does not contain the absolute location to the data directory anymore (but instead ${daisy.datadir}), so the restore wouldn't know where to create the data directory. This also means it is possible to restore the backup to another location then where it was originally.

If you were previously using older Daisy versions (< 1.5-M2), this of course requires that you updated the myconfig.xml to replace the absolute paths with ${daisy.datadir}, because the data is still restored to the locations specified in the myconfig.xml, the only difference is that ${daisy.datadir} is replaced with the path specified via the -d option.

9.14.2.4 Facultative options

9.14.2.4.1 -e <email-address>

Should exceptions occur during backup, notification emails can be sent to the specified address. These emails will contain the stacktrace of the exception. Do not forget to set the SMTP server using the next option.

9.14.2.4.2 -f <from-address>

This option allows users to specify the address sending exception notification emails.

9.14.2.4.3 -s <smtp-server>

Configures the smtp server to use to send emails in case of exceptions.

9.14.2.4.4 -a <entry-configuration-file>

If you want to backup other files you can create a configuration file to pass to the backup tool. In essence the configuration file contains a list of paths (files or directories) relative to a base directory. The content of these paths will be stored in a zip file in the backup directory.
Here is what the configuration file should look like:

```
<backup-entries>
  <backup-entry name="..." basedir="...">
    <paths>
      <path>...</path>
      ...
    </paths>
  </backup-entry>
  ...
</backup-entries>
```

And here is an example you might use to backup the Daisy Wiki files:

We don't simply backup the whole wikidata directory, as this would include the logs, which can become large. (a future feature might add an excludepath-option)

```
<?xml version="1.0" encoding="UTF-8"?>
<backup-entries>
  <backup-entry name="daisywiki" basedir="/home/daisy_user">
    <paths>
      <path>wikidata/books</path>
      <path>wikidata/bookstore</path>
      <path>wikidata/resources</path>
      <path>wikidata/sites</path>
      <path>wikidata/daisy.xconf</path>
      <path>wikidata/external-include-rules.xml</path>
    </paths>
  </backup-entry>
</backup-entries>
```

9.14.2.4.5 -q

Suppress the confirmation dialog prior to restoring an existing backup. This might be useful if you would like to restore a backup inside a batch file that has to run without manual intervention.

9.14.2.4.6 -v

Prints the version

9.14.2.4.7 -h

Shows the help

9.14.2.5 Postgresql notes

Supplying the Postgresql command line tools with a database password is not possible, therefore the Postgresql command line tools will prompt for a password when needed. However, there are some ways of bypassing the password prompt:

9.14.2.5.1 Password file

This password file is only used for the password prompt during a database dump. This means that it is only useful when you only want to automate backups. If you want to restore a database you will be prompted for a password twice, once for dropping the database and again for creating/restoring it. The password file is found here:

```
$HOME/.pgpass
```

This file MUST have the following permissions -rw------- (chmod 600)

It has this format :

```
host:port:databasename:username:password
```

Here is an example

```
localhost:*:daisyrepository:daisy:daisy
```

More information on the .pgpass file can be found here[10].

9.14.2.5.2 Trusted users

If you trust certain users they will not be prompted for passwords either. This is interesting when you couldn't be bothered to supply passwords to the command line during backup or restore. Information can be found here.[11]
Example :

```
#TYPE   DATABASE     USER    IP-ADDRESS        IP-MASK           METHOD
host    daisyrepository daisy   127.0.0.1         255.255.255.255   trust
host    template1       daisy   127.0.0.1         255.255.255.255   trust
host    all          all     127.0.0.1         255.255.255.255   md5
```

Be sure to place the users you trust before the users you do not (md5)

The database 'template1' must also be added because during the restore of a database a connection must be made with the Postgresql server and some database (the template1 database is sure to exist).

9.14.3 Restoring a backup

If you need to restore a backup on a blank system, you first need to:

- Install MySQL and create the `daisyrepository` and `activemq` databases and database users (see install instructions)
- Download the Daisy version corresponding to the backup, and extract it somewhere. Define the DAISY_HOME environment variable to point to this location.
- Install Java and let JAVA_HOME point to the location of your Java installation.

These items are described in more detail in the installation instructions.

What you don't need to do is running the installation scripts (daisy-repository-init, daisy-wiki-init, daisy-wiki-add-site), as these only serve to initialiase the database and configuration files, and when restoring a backup this is not needed since all this data and configuration comes from the backup.

Once this is done, use the backup tool with the `-r` option as described earlier.

9.15 JMX console

The Daisy repository server offers a *JMX* (Java Management Extensions) console through which certain things can be monitored or certain settings can be changed at runtime. In JMX, the components that provide this management access are called *MBeans*. The MBeans of the embedded ActiveMQ server are also visible through the same JMX console.

To access the JMX console, simply point your browser to

```
http://localhost:9264/
```

The JMX console listens by default only to requests from localhost. See below for how to change this.

You will be prompted for a password. Daisy automatically generates a password for accessing the JMX console during the Daisy installation. You can find this password in the `<daisydata dir>/conf/myconfig.xml` file, look for the following element:

```
<httpAdaptor port="9264" host="localhost" authenticationMethod="basic"
             username="daisyjmx" password="..."/>
```

Use the username and password mentioned on this element to log in. You can of course change these settings.

The repository server needs to be restarted for changes to the myconfig.xml to take effect.

If you want to be able to access the JMX console from non-localhost, simply change the value of the host attribute to the host name to which it should listen. To listen to all host names, use the value `0.0.0.0`.

As you will see in the myconfig.xml, there's also an xmlHttpAdaptor, listening on port 9265, which is exactly the same as the HTTP adaptor, except that it returns XML responses instead of HTML. This can be useful for programmatic access to JMX.

Daisy reuses the default JMX MBean server of the Java VM, and hence you can also use Java's `jconsole` tool to browse Daisy's MBeans. This requires passing the following option to the repository server VM: `-Dcom.sun.management.jmxremote`

9.16 Running parallel daisy instances

This is a short how to on running multiple daisies in parallel on one machine. By using different data directories it is possible to get multiple daisies up and running using only one set binaries.

9.16.1 Repository

9.16.1.1 Create a new data base (MySQL)

Create a new data base as described in the installation. Here is a summary.
Log in as root

```
mysql -uroot -pYourRootPassword
```

Create new data bases for the repository and activeMQ. In our example we'll just add the 'alt' suffix to the default data base names. We will also just reuse the same database users for these data bases.

```
CREATE DATABASE daisyrepositoryalt CHARACTER SET 'utf8';
GRANT ALL ON daisyrepositoryalt.* TO daisy@'%' IDENTIFIED BY 'daisy';
GRANT ALL ON daisyrepositoryalt.* TO daisy@localhost IDENTIFIED BY 'daisy';
CREATE DATABASE activemqalt CHARACTER SET 'utf8';
GRANT ALL ON activemqalt.* TO activemq@'%' IDENTIFIED BY 'activemq';
GRANT ALL ON activemqalt.* TO activemq@localhost IDENTIFIED BY 'activemq';
```

9.16.1.2 Create the data directory

9.16.1.2.1 Daisy repository initialisation

Go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-repository-init
```

Now here you must be carefull. After you choose your data base type, you will be asked to provide the data base URL, **DO NOT** use the default. Use this instead

```
jdbc:mysql://localhost/daisyrepositoryalt?characterEncoding=UTF-8
```

For the data base user, password, driver class, and driver location you can use the defaults. The same goes for the initial user.
You will now be asked if you wish to create the data base. When you are absolutely sure you typed in the data base URL for the new data base continue.

After the data base has been initialised you will be asked to choose the data directory location. This can be

```
/home/<daisyuser>/daisydata-alt
```

Now you are ready to pick a namespace. If you are planning to do imports and exports with other daisy instances on the machine then it would be best to pick a different namespace. ALT for example.

Now follow the instructions about mail configuration.

When you reach the embedded JMS part (activeMQ) make sure to use the right URL :

```
jdbc:mysql://localhost/activemqalt?characterEncoding=UTF-8
```

The rest of the steps you can use the defaults if you like.

9.16.1.2.2 Customizing the repository settings

Open the myconfig.xml in your favourite editor. This file is found in

```
/home/<daisyuser>/daisydata-alt/conf/myconfig.xml
```

Here is a list of changes you should make :

- Look for the term **'httpconnector'.** You will see the port 9263 mentioned there. Since your other daisy instance is using this port change this to a different value, 9563 for example.
- in the **mbeanserver** target you will find the httpAdaptor and xmlHttpAdaptor. Change their respective port attributes from 9264 and 9265 to 9564 9565.

9.16.1.2.3 Customizing activeMQ settings

Open activemq-conf.xml

```
/home/<daisyuser>/daisydata-alt/conf/activemq-conf.xml
```

Here you have to change the port activemq listens to. Look for **':61616'** and change this to something else like ':61656'.

9.16.1.2.4 Start the repository

Now you should be set to go. Try starting the repository. Go to <daisyhome> /repository-server/bin and run the startup script with your new data directory as argument.

```
./daisy-repository-server /home/<daisyuser>/daisydata-alt
```

9.16.2 Wiki

Again this is pretty much the same as the basic installation.

9.16.2.1 Initialize the wiki

Go to <DAISYHOME>/install and run

```
daisy-wiki-init
```

On the first question you will be asked to specify the repository. Here you can answer 'http://localhost:9563'. For the following questions on user and passowrd just use 'testuser' as username and password if you followed the defaults in the repository initialization that is.

9.16.2.2 Create the wiki data directory

9.16.2.2.1 Data directory initialization

Execute the following script

```
daisy-wikidata-init
```

When you are asked for a location of the data directory you might want to choose

```
/home/<daisyuser>/wikidata-alt
```

Next you will be prompted for the repository url again, answer 'http://localhost:9563'. For the following questions on user and password just use 'testuser' as username and password if you followed the defaults in the repository initialization that is.

Now you are asked for the repository data directory. In our example that is

```
/home/<daisyuser>/daisydata-alt
```

9.16.2.2.2 Wiki customization

Open de daisy.xconf file which is found in

```
/home/<daisyuser>/wikidata-alt/daisy.xconf
```

Now make the following changes :

- Look for 'repository-server-base-url'. Change this value from http://localhost:**9263** to http://localhost:**9563**.
- Change the jms server url. Look for 'tcp://localhost:**61616**' and change this to tcp://localhost:**61656**.

Now it is time to change the wiki port. First copy the jetty configuration to the data directory.

```
cp <DAISYHOME>/daisywiki/conf/jetty-daisywiki.xml /home/<daisyuser>/wikidata-alt
```

Open the jetty-daisywiki.xml file and change the port number. Look for **'jetty.port'** and change the value from '**8888**' to **'8588'.**

9.16.2.2.3 Create a site

Go to the directory `<DAISY_HOME>/install`, and execute:

```
daisy-wiki-add-site /home/<daisyuser>/wikidata-alt
```

First of all you will be prompted for the repository url again, answer 'http://localhost:9563'. For the following questions on user and password just use 'testuser' as username and password if you followed the defaults in the repository initialization that is.
Next you will be asked for a site name. You can answer 'alt-site' or something else if you please.

9.16.2.2.4 Running the wiki

If you plan on running the wiki using the startup scripts found in
<DAISYHOME>/daisywiki/bin/
Then you might run into some problems concerning temorary files. A workaround can be to copy the startup script and specify a different temporary directory.
First create a temporary directory. /var/tmp/alt for example

```
mkdir /var/tmp/alt
```

Create a new startup script

```
cp daisy-wiki <other-location>/daisy-wiki-alt
```

and change this line in the script

```
$JAVA_HOME/bin/java -Xmx128m -Djava.endorsed.dirs=$DAISYWIKI_HOME/endorsedlibs/
-Ddaisywiki.home=$DAISYWIKI_HOME -Dorg.mortbay.util.URI.charse
t=UTF-8 -Dfile.encoding=UTF-8 -Duser.language=en -Duser.country=US -Duser.variant=
-Ddaisywiki.data=$DAISYWIKI_DATA -Djava.awt.headless=true -
jar start.jar $CONFFILE
```

to

```
$JAVA_HOME/bin/java -Xmx128m -Djava.io.tmpdir=/var/tmp/alt
-Djava.endorsed.dirs=$DAISYWIKI_HOME/endorsedlibs/ -Ddaisywiki.home=$DAISYWIKI_HOME
-Dorg.mortbay.util.URI.charse
t=UTF-8 -Dfile.encoding=UTF-8 -Duser.language=en -Duser.country=US -Duser.variant=
-Ddaisywiki.data=$DAISYWIKI_DATA -Djava.awt.headless=true -
jar start.jar $CONFFILE
```

Start the wiki

```
daisy-wiki-alt  /home/<daisyuser>/wikidata-alt
```

Now everything should be up and running. Go to http://<server>:8588/daisy/ and look at your new daisy instance.

Notes

1. http://wrapper.tanukisoftware.org
2. http://wrapper.tanukisoftware.org/
3. http://wrapper.tanukisoftware.org/doc/english/donate.html
4. http://sourceforge.net/project/showfiles.php?group_id=39428&package_id=31591
5. http://jetty.mortbay.org/jetty/index.html
6. http://httpd.apache.org/docs-2.0/mod/mod_proxy.html
7. http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html
8. http://httpd.apache.org/docs-2.0/
9. http://wiki.apache.org/cocoon/ApacheModProxy
10. http://www.postgresql.org/docs/7.4/interactive/libpq-pgpass.html
11. http://www.postgresql.org/docs/7.4/interactive/client-authentication.html

10 Contributor/Committer tips

This section contains information useful for people working on the Daisy source code.

See also this related info in other places:

- Source code (page 43)
- Repository implementation notes (page 150)
- Daisy Wiki implementation notes (page 224)

10.1 Coding style

Without getting religious about coding style guidelines, it is pleasant and productive if all code follows more or less the same style. Basically, just do the same as the current sources, and follow the standard Java conventions (ClassNamesLikeThis, static finals in uppercase, etc). Opening braces are on the same line. Use some whitespace, e.g. write `x = 5 + 3`, not `x=5+3`.

One special point that requires attention: *use spaces for indentation, **do not use tabs***. For Java sources, we use 4 spaces, for XML 2 spaces.

For Javascript, please follow the Dojo style guide[1] (at the time of this writing, most Javascript in Daisy is a pretty unstructured set of functions, but as the amount of Javascript starts growing, we need to follow better practices).

10.2 Subversion configuration

Edit the file `~/.subversion/config`

Make sure the following line is not commented out:

```
enable-auto-props = yes
```

In the section [auto-props], add the following entries:

```
*.js = svn:eol-style=native
*.xml = svn:eol-style=native
*.java = svn:eol-style=native
*.txt = svn:eol-style=native
*.xconf = svn:eol-style=native
*.xweb = svn:eol-style=native
*.xmap = svn:eol-style=native
*.properties = svn:eol-style=native
*.css = svn:eol-style=native
```

```
*.xsl = svn:eol-style=native
*.xsd = svn:eol-style=native
*.dtd = svn:eol-style=native
*.ent = svn:eol-style=native
*.nsh = svn:eol-style=native
*.nsi = svn:eol-style=native
*.ini = svn:eol-style=native
*.conf = svn:eol-style=native
```

10.3 Submitting a patch

Always start from an SVN checkout to edit sources. This can be the trunk, where the main development is happening, or a maintenance branch.

See the information on getting the sources (page 43).

When you want to contribute changes as a patch, go to the root of the Daisy source tree, and enter:

```
svn diff > mypatch.txt
```

This will create a file called mypatch.txt containing all the changes you made to the source tree. You can make a diff of just some files by listing them after the diff command.

- It is recommended to have a look in the patch file to see if it doesn't contain irrelevant changes.
- Before doing a diff, do an "svn update" to make sure you have the latest code
- Generally it is a good idea to discuss changes beforehand on the mailing list.
- See the SVN book[2] to learn more about subversion.

If you have created new files that are not yet in the subversion repository, you need to add them first using svn add, before doing svn diff:

```
svn add path-to-new-file
```

You can then add your patch as an issue in Jira[3]. A good description/motivation about your patch will help to get it applied.

Thanks in advance for your contribution!

10.4 Maintaining change logs

(for committers)

When committing a feature change, make sure to mention it on the changes page for the current release (which is part of the documentation of that release). Same holds for commits which involve backwards incompatible changes or changes that require special action when updating.

When committing a change which requires people to make database changes, configuration changes or some other changes to their setup, please add them to this page on the community wiki (page 0) (besides notifying the mailing list of course).

10.5 Artifacter tool

10.5.1 Introduction

Daisy makes intensive use of the artifact (page 0) concept. Both Maven[4] and the Daisy Runtime (page 141) make use of these artifacts.

Since there are a lot of different Maven POMs and Daisy Runtime classloader definitions, it is useful to have some help in managing these. For example when you want to find out which projects use commons-collections or update all of them to a newer version. Or you want to augment the version number of all Daisy artifacts.

Rather than doing this manually, which is boring and error-prone, we have a tool called the "artifacter" which does this.

10.5.2 General usage instructions

First compile the artifacter tool:

<daisy-src> refers to the root of the Daisy source tree

```
cd <daisy-src>
cd tools/artifacter
maven jar:jar
```

Then the typical usage is:

```
cd <daisy-src>

./tools/artifacter/target/artifacter-dev <options>
```

You should usually invoke the tool from the root of the Daisy source tree, since it will work on all files found recursively from the current working directory.

10.5.3 Finding out which projects use a certain artifact

This lists all files where a certain artifact is used.

Note that the argument of the -f parameter is <groupId>:<artifactId>

```
./tools/artifacter/target/artifacter-dev -f daisy:daisy-repository-api
```

10.5.4 Upgrading a dependency

For example, let's move to a new version of Jetty:

```
./tools/artifacter/target/artifacter-dev -u jetty:org.mortbay.jetty:16.2.3
```

10.5.5 Upgrading a project version number

Within Daisy, this is only useful for upgrading the Daisy version number.

It upgrades version numbers not only of dependencies, but also of the project definitions themselves.

It works on all artifacts within a certain group.

Example usage:

```
./tools/artifacter/target/artifacter-dev -g daisy:2.8
```

10.5.6 Renaming an artifact

Less common, but it can happen (this was mainly used in the early Daisy days).

```
./tools/artifacter/target/artifacter-dev
    -r -o daisy:daisy-repository-api -n daisy:daisy-repository-api2
```

10.5.7 Creating a repository of all dependencies

This creates a Maven-style repository containing all dependencies from the runtime classloader definitions. This is used by the binary distribution build script.

```
./tools/artifacter/target/artifacter-dev -c -s <source-repo> -d <dest-repo>
```

The <source-repo> is typically ~/.maven/repository

The <dest-repo> is typically a new empty directory.

10.5.8 Copy project dependencies

This copies the dependencies of a certain Maven project to a Maven-style local repository. This is used by the binary distribution build script.

```
./tools/artifacter/target/artifacter-dev -l -s <source-repo> -d <dest-repo>
```

10.5.9 Copy artifact

This copies the artifact build by a certain Maven project to a Maven-style local repository This is used by the binary distribution build script.

```
./tools/artifacter/target/artifacter-dev -a -s <source-repo> -d <dest-repo>
```

Notes

1. http://dojotoolkit.org/docs/js_style_guide.html
2. http://svnbook.red-bean.com/
3. http://issues.cocoondev.org/
4. http://maven.apache.org

11 FAQ

See the knowledgebase[1] for FAQs.

Notes

1. /kb/

12 Glossary

Definitions in the glossary:

Name
Artifact
Artifact
myconfig.xml
myconfig.xml
Repository data directory
Repository data directory