

# Daisy features overview

---

## Introduction

A content management system (CMS), as the name implies, is concerned with managing content (information). This includes things like basic storage and retrieval, authoring, browsing and searching content, versioning, access control, workflow and publishing. Daisy makes content management accessible to everyone, by combining a lot of valuable functionality in one easy-to-use package.

Daisy is suited for a variety of CMS applications, such as websites, knowledge management, intranets, product documentation, ...

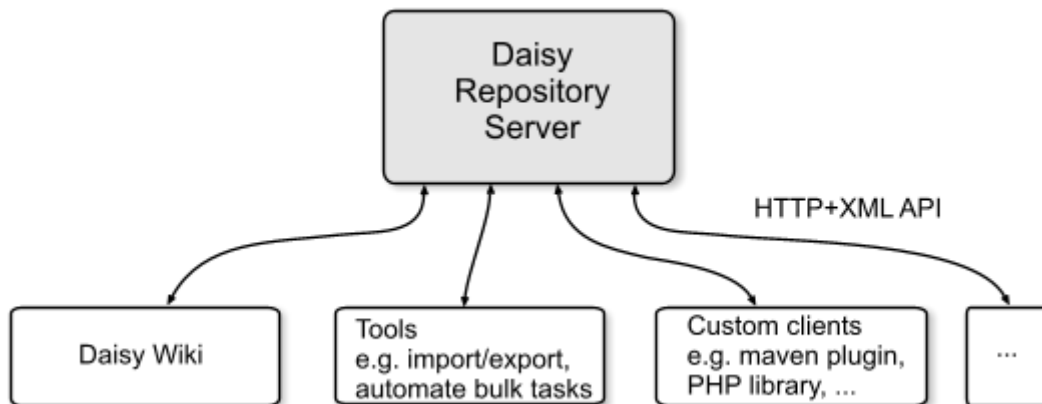
## Documents

The core entity managed by Daisy is called a document. Compared to a simple file on normal filesystem, a Daisy document has a more detailed structure: it can contain multiple binary data parts, and a variety of fields, which are simple values like strings, dates, numbers, or links to other documents. Compared to a record in a typical RDBMS database table, a Daisy document is a heavier entity offering more functionality and behavior, such as versioning, change notifications and access control.

It is possible to store any kind of content (binary data) in Daisy documents, ranging from text over pictures to movies, i.e. any sort of binary-encoded information, even if it's very big.

Daisy however comes with built-in support for an XML-based subset of HTML, for which it can do all sorts of interesting things. It can extract link information from these documents, it supports a flexible and powerful publishing model for them (with support for inclusions, embedded queries, ...), it supports browser-based rich text editing of them, it supports combining many of these documents in order to publish a polished book in different formats.

## Content repository separate from front-end



The Daisy project is split into two parts. First there is the **content repository** which provides the core content management tools, i.e. the pure content management without graphical user interface (GUI). The repository server runs as a standalone process and is fully accessible through a HTTP+XML based interface (ReST-style). The repository server has a Java API which is both usable inside the repository server and remotely, in the last case it will transparently communicate over the HTTP+XML API.

Next to this, there is the **Daisy Wiki**, which is a general-purpose web-based GUI on top of the content repository. It provides publishing, editing, and administrative features. The Daisy Wiki is [evolving towards] a generic frontend platform which can be customized and extended to tune it to your own needs. Next to this, it is possible to write your own client applications that communicate with the repository server. You could choose not to use the Daisy Wiki at all, if it does not fit your needs.

## Open source

Daisy is available for free with source code included. The sources are released under the business-friendly Apache license. Daisy has frequent releases, an active community, and development happens openly and transparently.

Daisy is a Java-based application, and is based on the work of many great open source packages, without which Daisy would not have been possible. All third-party libraries or products we redistribute are unmodified (unforked) copies.

The remainder of this document provides a more detailed listing of the functionality offered by the repository server and the Daisy Wiki.

## The document repository

### Documents

Document are the core entities in Daisy, the main purpose of Daisy is managing documents.

A document can consist of multiple content *parts* and *fields*, a *document type* defines what parts and fields a document should have.

*Fields* can be of different data types, such as string, date, decimal, boolean, link, ... A field can be a multi-value field, in which case it can contain an (ordered) set of values. A field can be a hierarchical field, in which case its value is a hierarchical path, for example something like 'Category / Sub category / Sub sub category'. A field can be multi-value and hierarchical at the same time. Field values can optionally be selected from a selection list.

*Parts* can contain arbitrary binary data, but the document type can limit the allowed mime types. So a document (or more correctly a part of a document) could contain XML, an image, a PDF document, plain text, ... Part upload and download is handled in a streaming manner, so the size of parts is only limited by the available space on your filesystem (and for uploading, a configurable upload limit).

Sometimes people call the fields 'metadata', however they can be considered both as data or metadata depending on how one looks at it. A document can have only fields and no parts, or only parts and no fields, or even no parts and fields at all (not very useful).

As mentioned, fields can be of various data types. One special case among the supported types are the link-type fields. *Link-type fields* allow to connect documents together in a structured way, for example a document about a wine can be connected to a document describing its origin region. The query language has a dereference operator which allows to walk through these links in queries. This makes that the Daisy repository does not just manage documents with some meta data attached, but allows to do things for which you would otherwise need a relational database (though Daisy is by no means a substitute for those!).

### Flat structure, no default hierarchy

The repository has no hierarchical directory structure, all documents are stored in one "big bag".

A document is identified by a unique ID (an ever-increasing sequence number starting at 1), and has a name (which does not need to be unique).

The lack of directory structure avoids that one needs to think about where to store a document when creating it. Documents can be displayed and re-used in various contexts. For example, the Daisy Wiki allows to define multiple hierarchical navigation trees on the same repository.

## Versioning

Every time a document is saved, a new revision is created, allowing to keep track of who changed what and when. The basic versioning model is linear, though through variants (see later on) branches can be created.

Each version can be either in the state 'published' or in the state 'draft'. The most recent version which has the state published is the 'live' version, this is the version that is usually displayed by default.

## Variants

A document can exist in multiple variants. Each variant has its own version history. Each variant is basically a different copy of the same document.

There are two types of variants: branches and languages. Language variants are useful to maintain documents in multiple languages, while branch variants allow parallel maintenance of different 'versions' of a document. Branches are often useful for software documentation, for example Daisy's own documentation has a branch for each major version.

## Locking

It is possible to take exclusive locks on documents for a limited or unlimited time. Checking for concurrent modifications (optimistic locking) happens automatically.

## Retiring (archiving) and deletion

A document can be marked as 'retired', which virtually deletes it; it won't show up in query results by default. It is however still possible to retrieve the document, and to un-retire it. A document can also be deleted permanently, removing all traces of it.

## Collections

Documents can be combined in so-called collections. Collections are sets of the documents. A document can belong to multiple collections, in other words, collections can overlap.

As such, a collection can also be seen as a form of built-in metadata for grouping documents.

## Querying

A high-level query language, not unlike SQL (select ... where ... order by ...), provides a flexible and simple way to query the repository for documents. The query language allows to search on built-in document metadata and custom document fields, to perform full text searches, and some other special conditions.

The query results only contains documents the user is allowed to see (more in general, the repository server never allows a user access to something it isn't allowed to access -- quite normal of course).

Normally queries search in the 'live' version of a document, however through a query option it is also possible to search in the last version of a document.

Behind the scenes, queries are translated to SQL queries and queries on a full text index. Queries in Daisy's query language are quite simple compared to the SQL generated from them.

Documents are automatically full-text indexed (Jakarta Lucene based). Currently supported formats include plain text, XML, PDF (through PDFBox), MS-Word, Excel and Powerpoint (through Jakarta POI), and OpenOffice Writer. Full-text indexing happens asynchronously (in the background).

Next to normal queries, “faceted queries” are also supported. Basically faceted queries add the distinct values and their counts of a number of metadata fields to the query results so that they can be presented to the user for faceted navigation. The Daisy Wiki includes a configurable and customizable frontend for this.

## Repository namespaces

Each Daisy repository server can be assigned a namespace. A namespace avoids conflicts between the identification of documents in different repositories, enabling transfer of documents between repositories while keeping their identity intact.

Daisy includes import and export tools to move documents between repositories. The export tool exports selected content of a repository to an intermediate format, and the import tool imports it into another repository. The intermediate format is documented and could be created or used by other tools as well.

## Centralized access control

Many systems allow to attach access control settings to individual documents or directories. Since Daisy doesn't have directories, and because access control settings for individual documents are a management nightmare, Daisy instead uses a global access control list (ACL) which allows to specify the access rules for sets of documents by selecting those documents based on expressions.

This allows for example to define access control rules for all documents of a certain type, or for all documents in a certain collection.

Read access can be defined not only for documents as a whole, but also on a sub-document level. For example, you can use this to limit access to only metadata, and reserve full access to users with certain roles.

## Translation management

Earlier, we mentioned that a document can exist in multiple language variants. Daisy has built-in features to keep track of the translation status of these language variants with respect to the reference language of the document. The query language supports searching for out-of-date or missing translations. The import/export tools supports a special mode suited for delivering content to external translation agencies, which typically have their own optimized toolchain (with translation memories etc.).

## APIs

Since the Daisy repository server was from the start intended to be run as a standalone process, it offers solid APIs for remote communication.

The repository server can be accessed by means of a HTTP+XML protocol (ReST-style), thus providing language and platform neutral access. The documentation of the HTTP interface includes examples on how the repository can be updated using command-line tools like wget and curl. The HTTP interface is not auto-generated from the underlying repository implementation, but separately designed, thus working with meaningful URLs and XML messages.

A high-level, easy to use Java API, is available both as an "in-JVM" implementation for embedded scenarios or services running in the Daisy server VM, as well as an implementation that communicates transparently using the HTTP+XML protocol, using a sensible granularity for remote calls. It can also be handy to use this API from Java-hosted scripting languages.

Using either the HTTP or Java API it is easy to automate various tasks (such as bulk updates), as well as to develop custom client applications.

## Events

For various repository events, such as document creation and update, events are broadcasted via JMS. Daisy includes an embedded ActiveMQ broker for this. The content of the JMS events are XML messages.

Internally, the JMS events are used for updating the full-text index, for triggering the sending of notification mails, and for clearing remote caches.

Basically all operations on the repository server which change the persistent state of the repository cause a JMS event to be produced. Hence logging all JMS events gives a full audit log of all updates that happened to the repository.

## Extensions

Repository extensions can add additional services to the repository server, which don't really need to be in the core repository. The following sections list some extensions provided with Daisy.

### Email notifier

This extension sends e-mail notifications when certain events happen in the repository server. It also manages the subscription to these notifications. Users can subscribe to individual documents, collections of documents or all documents.

### Publisher

This extension allows to retrieve aggregated and prepared data for rendering, this to minimize the amount of remote calls that one needs to make to the repository, and to make some core publishing functionality reusable. This component is used by the Daisy Wiki to do its document publishing, but other applications have also made use of it, such as the Forrest and Maven plugins, or the PHP client library.

### Navigation manager

This extension provides functionality for generating hierarchical navigation trees. A navigation tree definition can contain manually defined structures but also queries which generate navigation hierarchies. Navigation tree nodes are filtered according to the access rights of users. See the Daisy Wiki for more details on this component.

### Document task manager

This extension allows for the reliable background execution of a certain task across a (potentially large) set of documents. Execution progress is tracked persistently in the database. Ideal for applying bulk changes to documents.

### Workflow

This extension provides flexible workflow functionality, internally based on jBPM.

## Other features

### Image handling

The Daisy repository contains a component which can generate image thumbnails, extract EXIF data to document fields, and auto-rotate images based on EXIF information. This component is implemented in the form of a “pre-save hook”, and supports a flexible configuration to handle different document types.

### Link extraction

The repository keeps track of the links between documents. To do this, it supports extracting links from a number of formats, especially the “Daisy HTML” format. Adding new link extractors is possible. Link extraction allows to quickly search for document referers, i.e. documents linking to a document.

### Summary extraction

Daisy can extract and store a short summary from documents. This can be useful to display in search results. Currently summary extraction works by getting the first ~300 characters from the first Daisy HTML part of the document.

### Comments

Daisy supports a simple document commenting feature. It is expected that in the future, the commenting feature will be moved outside of the core repository into an extension component, and then be further enhanced to support hierarchical comment threads etc.

## Administration features

### Authentication

If desired, authentication of users can be performed by external or custom systems. Support for LDAP and NTLM is included out-of-the-box. It is possible to auto-create users in Daisy upon first login, if they exist in the external authentication system.

### Backup

The Daisy repository allows to take a global write-lock for the purpose of creating a consistent backup without taking the repository server down. A tool for performing backups is included with Daisy.

### JMX console

A JMX console allows some monitoring and maintenance operations, such as optimization or rebuilding of the fulltext index, monitoring memory usage, document cache size, or database connection pool status.

## Data storage

Repository data is stored in a relational database. Our main development happens on MySQL/InnoDB, but the provisions are there to add support for new databases. The part content is stored in normal files on the file system. Since people often wonder about why this is: it removes load from the database, and many databases and/or JDBC drivers don't support streaming handling of part data, limiting both file sizes and causing serious performance degradation.

The usage of these familiar, open technologies, combined with the fact that the Daisy Wiki frontend stores plain HTML, and the easy-to-use APIs, makes that your valuable content is easily accessible with minimal “vendor” lock-in.

## The Daisy Wiki frontend

The Daisy frontend is called the *Daisy Wiki* because, just like wikis, it provides a mixed browsing/editing environment with a low entry barrier, hence providing a writeable-web experience. However, there are some big differences with the original wikis. The Daisy Wiki has wysiwyg editing, has a powerful navigation component, and inherits all the features of the underlying Daisy repository such as different document types and powerful querying.

### Dynamic document editor with wysiwyg HTML editing

As mentioned earlier, a document in Daisy can have multiple parts and fields as defined in a document type. The document editor of the Daisy Wiki automatically adjusts itself according to the document type, so no custom development is involved in this.

For “Daisy HTML” parts, a wysiwyg (what-you-see-is-what-you-get) HTML editor is used. This editor is supported on current Internet Explorer and Mozilla/Firefox (gecko) browsers, with fallback to a plain textarea (HTML source editing) on other browsers.

For other parts, an upload control is used, but custom part editors can be implemented, as we have done for the navigation tree editor (see further).

The HTML editor does not allow arbitrary HTML. Instead, it is limited to a small, structural subset of HTML, so that the content is future-safe, output medium independent, secure and easily transformable. It is possible to have special paragraph types such as 'note' or 'warning'. The stored HTML is always well-formed XML, with a nice source-layout formatting. Thanks to a powerful (server-side) cleanup engine, the stored HTML is exactly the same whether edited with IE or Mozilla, allowing to do source-based diffs.

Images can be easily inserted in the HTML editor, by uploading new images or browsing existing images. Images themselves are separate documents in the repository, so they can also have versions, metadata, access control, etc.

Links to other documents can be created through a powerful dialog that allows you to browse for the target document to link to.

When starting to edit a document, an exclusive lock is automatically taken on the document, with an expire time of 15 minutes, and the lock is automatically refreshed while the document editor is open.

For editing the fields, the document editor also dynamically creates suited editors according to the document type. Just as for parts, custom field editors can be implemented.

### Document publishing

The Daisy Wiki has a quite powerful document publishing model. Some of its features are:

- document includes, thus the inclusion of one document into another. Includes are handled recursively. Next to documents, content fetched from arbitrary external sources can also be included. For example, 'http:' or 'file:', but also 'cocoon:' which allows to call Cocoon-based extensions (see the section on extensions).
- embedding Daisy-queries in pages, with the possibility to customize the rendering of the query results (thus it is not necessarily a plain table).

- custom document styling (XSLT-based), mostly done depending on the type of the document. This document styling is applied regardless of whether the document is displayed stand-alone, or when included inside another document, or published in some other context (such as RSS feeds or custom aggregations).
- variable substitution. Documents can contain variables which are substituted during publishing. A typical use-case is to avoid hard-coding product names in documentation, since the product might be marketed with different names.
- when rendering a document, it is possible to aggregate additional information with it such as queries, content from related documents, navigation trees, ... Completely custom pages are possible through extensions.
- PDF publishing (using XSL-FO through Apache FOP), with all the same features as the HTML publishing.
- The frontend currently doesn't perform any caching of rendered documents, all documents are published dynamically, since this depends on the access rights of the current user. Nonetheless, the document rendering architecture has been designed for performance, and is very fast.

## Hierarchical navigation

Navigation trees can be defined to provide the user with hierarchical navigation. As many navigation trees as you want can be created.

Navigation trees are defined as XML and stored in the repository as documents, thus access control, versioning etc. applies. One navigation tree can import another one. The nodes in the navigation tree can be listed explicitly, but also dynamically inserted using queries.

When a navigation tree is generated, the nodes are filtered according to the access control rules for the requesting user. Navigation trees can be requested in "full" or "contextualized", this last one meaning that only the nodes going to a certain document are expanded. The navigation trees are generated as XML, the visual rendering is up to XSL stylesheets.

A navigation tree editor widget allows easy editing of the navigation trees without knowledge of XML. The navigation tree editor works entirely client-side, without annoying server-side roundtrips to move nodes around, and with full undo support.

## URL space

The URL space in the Daisy Wiki is driven by the hierarchical navigation tree. That is, the path in the URL corresponds to the hierarchical position of the document in the navigation tree (if any). By default, the path items are simply the document IDs, though it is possible to assign custom names to these to have nice readable URLs. However, keeping the document ID in the URL path has the advantage that it allows to find back the document when the navigation tree has been reorganized.

## Searching

The Daisy Wiki provides various search-related pages:

- fulltext search, including display of matching content extracts
- a page to search using Daisy's query language
- faceted navigation (browsing/filtering using the metadata of documents)
- recent changes
- display of the referrers of a document (incoming links)

Behind the scenes, all these pages are implemented using Daisy's query language, they just add some specific query form and appropriate rendering of the results.

## Sites

Multiple sites can be defined. Each site is a different view on top of the same Daisy repository. Each site can have a different navigation tree, home page, skin and is associated with a default collection. Newly created documents are automatically added to this default collection, and searches are limited to this default collection (unless requested otherwise).

## Version management

Version-management features include:

- viewing the list of available versions
- viewing the content of the various versions
- comparing (diffing) versions
- reverting to a previous version
- changing the state (between publish or draft) of versions

The Daisy Wiki as a whole can be switched between 'LIVE' or 'STAGING' mode. In live mode, the live versions of documents are displayed by default. Documents which have no live version yet are not displayed. When switching to staging mode, the last version of all documents is displayed by default. This also affects the execution of queries and the generation of the navigation tree. The staging mode basically shows what the site would look like if the last version of all documents would become live.

The version comparison supports both the usual text-based diffs (which compares the HTML source), as well as a visual diff showing the rendered document with indications of added and removed text and formatting.

## Skinning

The look of the Daisy Wiki can be customized through so-called skins.

A skin consists of various resources such as CSS files, images, possibly some javascript, and XSLT stylesheets. Basic customizations can be done by adjusting the CSS, changing the XSLT templates allows to change the layout completely.

The skinning mechanism supports inheriting resources between skins. This means a custom skin can be based upon an existing skin, only the resources which you want to change need to be part of your skin.

## Extensions

Daisy Wiki extensions provide a hook for adding custom developed pages to the Daisy Wiki. These can be literally anything, such as custom document renderings, custom search pages, forms, or things that have nothing at all to do with Daisy.

Daisy contains a number of example extensions, e.g. RSS feeds are implemented in the form of an extension. The Daisy community Wiki also shows a number of extensions, such as a calendar-view of documents.

Extensions are built on top of Apache Cocoon (see next item). For people familiar with Cocoon, an extension is basically a mounted sub-sitemap, and can make use of various Daisy Wiki services.

## Apache Cocoon web-framework

The Daisy Wiki is built on the op Apache Cocoon, an XML-oriented web publishing and application framework, ideally suited for this type of application. Besides being useful for the document publishing of the Wiki, it also provides a number of attractive web-development features:

- A so-called “sitemap” which is a flexible mechanism for specifying how an incoming request should be handled.
- Javascript-based, continuations-supporting flow control.
- A high-level form framework, handling the complexities of showing a HTML form until it is correctly completed, with support for strong data typing, validation, event handling and Ajax.

## Other things

### Internationalization

The complete frontend is localizable through resource bundles. Multiple languages are included.

### User self-registration

User self-registration (with the possibility to configure which roles are assigned to users after self-registration) and password reminder.

### Document basket

Documents can be collected in a 'document basket'. Using the documents collected in the basket, a user can then perform some action on them, such as getting an aggregated HTML or PDF page, or performing a document task upon them.

## Book publishing

The book publishing component allows for the generation of professionally formatted books with table of contents, section numbering, cross-referencing, footnotes, index, lists of images and tables, etc. It goes much further than simply concatenating some pages. Headers in documents are shifted based on their hierarchical position within the book, and when doing a chunked-HTML publication, the chunking is performed irrespective of the boundaries of the original documents.

Books can be produced in a variety of formats, HTML and PDF publications are included by default.

Books are generated as a batch process, the end-result is a static book publication that could be put on a static web server or a compact disc.

## Management interface

The Daisy Wiki includes GUI screens for managing users, the repository schema, the access control, the workflow, etc.

## General

### Unicode support

Daisy uses unicode everywhere, both in the repository server as well as the Daisy Wiki.

## Further pointers

- [Download Daisy](#)<sup>1</sup>
- [Daisy documentation](#)<sup>2</sup>
- [Online Daisy demo](#)<sup>3</sup>
- [Daisy movie-tutorial](#)<sup>4</sup>
- [Daisy mailing list](#)<sup>5</sup>