

Feature overview (technical)

The Daisy project encompasses two major parts: a content repository and a web-based, wiki-like frontend (a mixed browsing/editing environment). If you have different frontend needs than those covered by the standard Daisy frontend, you can still benefit hugely from building upon its repository part. Or you could reuse the Daisy Wiki as editing environment, and build a custom publishing frontend.

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

The document repository

Some of the main features of the document repository are:

- Storage and retrieval of documents.
- Documents can consist of multiple content *parts* and *fields*, *document types* define what parts and fields a document should have. Fields can be of different data types (string, date, decimal, boolean, ...) and can have a list of values to choose from. Multi-value fields are also possible. 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).
- Versioning of the content parts and fields. Each version can have a state of 'published' or 'draft'. The most recent version which has the state published is the 'live' version, this is the version that is displayed by default (depends on the behaviour of the frontend application of course).
- Branch variants and language variants: each document can exist in multiple variants. Each variant has its own version history. Language variants are useful to maintain documents in different languages, while branch variants allow parallel maintenance of different 'versions' of a document (which is for example useful for software documentation).
- Document comments.
- Documents can be marked as 'retired', which makes them appear as deleted, they won't show up unless explicitly requested. Documents can also be deleted permanently.
- The repository doesn't care much what kind of data is stored in its parts, but if it is "HTML-as-well-formed-XML", some additional features are provided:
 - link-extraction is performed, which allows to search for referers of a document.
 - a summary (first 300 characters) is extracted to display in search results
 - (these features could potentially be supported for other formats also)
- All documents are stored in one "big bag", there are no directories. Each 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). Hierarchical structure is provided by the frontend by the possibility to create hierarchical navigation trees.
- Documents can be combined in so-called *collections*. Collections are sets of the documents. One document can belong to multiple collections, in other words, collections can overlap.
- It is possible to take exclusive locks on documents for a limited or unlimited time. Checking for concurrent modifications (optimistic locking) happens automatically.

- Documents are automatically full-text indexed (Jakarta Lucene based). Currently supports plain text, XML, PDF (through PDFBox), MS-Word, Excel and Powerpoint (through Jakarta POI), and OpenOffice Writer.
- 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 (to offload the database). The usage of these familiar, open technologies, combined with the fact that the Daisy Wiki frontend stores plain HTML, makes that your valuable content is easily accessible with minimal "vendor" lock-in.
- A high-level, SQL-like query language provides flexible querying without knowing the details of the underlying SQL database schema. The query language also allows to combine full-text (Lucene) and metadata (SQL) searches. Search results are filtered to only contain documents the user is allowed to access (see also access control). The content of parts (if HTML-as-well-formed-XML) can also be selected as part of a query, which is useful to retrieve eg the content of an "abstract" part of a set of documents.
- Access control: instead of attaching an ACL to each individual document, there is a global 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.
- The full functionality of the repository is available via an HTTP+XML protocol, thus providing language and platform independent access. The documentation of the HTTP interface includes examples on how the repository can be updated using command-line tools like wget and curl.
- A high-level, easy to use Java API, 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 either the HTTP or Java API it is easy to automate various tasks, such as mass-import or export of data.
- For various repository events, such as document creation and update, events are broadcasted via JMS (currently we include OpenJMS). The content of the events are XML messages. Internally, this is used for updating the full-text index, notification-mail sending and clearing of remote caches. Logging all JMS events gives a full audit log of all updates that happened to the repository.
- 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.
- Repository extensions can provide additional services, included are:
 - an email-notification component (which also includes the management of the subscriptions), allowing subscribing to individual documents, collections of documents or all documents.
 - a *Navigation Manager* component that supports the generation of hierarchical navigation trees out of the repository.
 - a *Publisher* component, which allows to retrieve aggregated and prepared data for rendering, this to minimize the amount of remote calls that one needs to make from the frontend, and to make some basic publishing work reusable.
 - a *Document Task Manager* component, which allows the reliable background execution of a certain task(s) across a (potentially large) set of documents.
- 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.

The Daisy Wiki frontend

The frontend is called the *Daisy Wiki* because, just like wikis, it provides a mixed browsing/editing environment with a low entry barrier. However, it also differs hugely from the original wikis, in that it uses 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.

Here are some of the main features and differentiators:

- wysiwyg HTML editing
 - supports recent Internet Explorer and Mozilla/Firefox (gecko) browsers, with fallback to a textarea on other browsers.
 - arbitrary HTML is not allowed. 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, and nicely layed-out. 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.
 - insertion of images by browsing the repository or upload of new images (images are also stored as documents in the repository, so can also be versioned, have metadata, access control, etc)
 - easy insertion document links by searching for a document
 - a heartbeat keeps the session alive while editing
 - an exclusive lock is automatically taken on the document, with an expire time of 15 minutes, and the lock is automatically refreshed by the heartbeat
 - editing screens are built dynamically for the document type of the document being edited.
- Version overview page, from which the state of versions can be changed (between published and draft), and diffs can be requested.
- Nice version diffs, including highlighting of actual changes in changed lines (ignoring re-wrapping).
- Support for includes, i.e. the inclusion of one document in the other (includes are handled recursively).
- Support for embedding queries in pages, with the possibility to customise the rendering of the query results (thus it isn't necessarily a plain table).
- Document type specific styling (XSLT-based). This also works nicely combined with includes: each included document will be styled with its own stylesheet depending on its document type.
- A hierarchical navigation tree manager. 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 (for authoring them, read access is public), 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 tree manager produces 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 (Mozilla/Firefox and Internet Explorer), without annoying server-side roundtrips to move nodes around, and full undo support.
- PDF publishing (using Apache FOP), with all the same features as the HTML publishing, thus also document type specific styling.
- Multiple search pages:
 - fulltext search

- a page to search using Daisy's query language
- faceted navigation (browsing by the metadata of documents)
- display of referers (incoming links)
- Multiple-site support, allowing to have multiple perspectives on top of the same daisy repository. Each site can have a different navigation tree, 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).
- XSLT-based skinning/layouting. Skins are configurable on a per-site basis.
- User self-registration (with the possibility to configure which roles are assigned to users after self-registration) and password reminder.
- Comments can be added to documents.
- Internationalization: the whole front-end is localizable through resource bundles.
- Management pages for managing:
 - the repository schema (the document types)
 - the users
 - the collections
 - the variants
 - access control
- The frontend currently doesn't perform any caching of rendered pages, all pages are published dynamically, since this depends on the access rights of the current user. Page rendering is however quite performant. For publishing of high-traffic, public (all access as the same user), read-only sites, a custom publishing application can be developed.
- The frontend is built on top of Apache Cocoon, an XML-oriented web publishing and application framework, ideally suited for this type of application.
- Extension hooks, making it possible to develop new pages that integrate into the Daisy Wiki, or to generate content that can be dynamically included into documents. The development of these extensions is based on the Cocoon framework, combined with the high-level Daisy repository API. A number of example extensions is included, such as RSS feeds. These extensions can be build without needing any Java development (but could, if you want).
- **Book publishing:** a component which allows for the generation of nicely formatted books with table of contents, section numbering, cross-referencing, footnotes and index. It goes much further than simply concatenating some pages.