

# Import tool

---

The import tool imports data conforming to the Daisy [import/export format](#)<sup>1</sup> 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](#)<sup>2</sup>.

To see all command line arguments supported by the import tool, execute:

```
daisy-import -h
```

## 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.

## 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.

## 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](#)<sup>3</sup>. Note that if any queries are specified in the document set file, these queries are executed on the target repository.

## 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`:

- 
1. [daisy:332-cd](#) (Import/export format)
  2. [daisy:333-cd](#) (Import/export introduction)
  3. [daisy:335-cd](#) (Export tool)

```
<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.

## 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
```

## 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.   |
| fullStackTracesOfFailures | 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. |
| changeComment          | Version change comment for new versions. Default is "Updated by import".  |
| changeType             | Version change type for new versions. Should be major or minor. Default is major.   |
| storeReferenceLanguage | Sets whether the reference language should be imported. Default true. When false, the reference language won't be set or changed.   |

## 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 fields as mentioned earlier, it also allows 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.

## 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 your class, it needs to be available 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
```

## 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.

## 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`.