Legacy methods for re-indexing content

Please note, that as of DSpace 4.0, the Solr-based Discovery search is on by the default in both JSPUI and XMLUI. This page describes the older Lucene-based search and DBMS browse indices. Neither the DBMS browse tables nor the Lucene search indices are used anymore (unless you explicitly disable SolrBrowseDAO and enable search artifacts). This page was previously called ReIndexing Content with the old legacy providers (DBMS for Browse or Lucene for Search)

Overview

DSpace offers two options to index content for Browsing & Searching:

1. Faceted/Filtered Search & Browse (via Solr & DSpace Discovery) - **enabled** by default since DSpace 4.0
2. Traditional Browse & Search (via Lucene & Database tables) - this is **disabled** by default

This particular page only describes the "Traditional Browse & Search" indexing processes. For more information on Faceted/Filtered Browse & Search, please see DSpace Discovery, in particular Discovery Solr Index Maintenance .

Re-Enabling the legacy Lucene Search and/or DBMS Browse providers

TO BE COMPLETED

TODO: also add the DB-backed itemcounter here (?)

DBMS Browse Providers

If a DAOs configuration is not provided the system will use the SOLR Browse Engine

Configure the browse engine to use PostgreSQL

This option enables the browse engine to store its indexes in PostgreSQL database tables. All browsing is then performed via queries to those database tables. This is the traditional browsing option for users of PostgreSQL. The configuration is as follows:

```
browseDAO.class = org.dspace.browse.BrowseDAOPostgres
browseCreateDAO.class = org.dspace.browse.BrowseCreateDAOPostgres
```

Configure the browse engine to use Oracle

This option enables the browse engine to store its indexes in Oracle database tables. All browsing is then performed via queries to those database tables. This is the traditional browsing option for users of Oracle. The configuration is as follows:

```
browseDAO.class = org.dspace.browse.BrowseDAOOracle
browseCreateDAO.class = org.dspace.browse.BrowseCreateDAOOracle
```

Creating the Browse & Search Indexes

To create (or recreate) all the various browse/search indexes that you define as described in this page there are a variety of options available to you. You can see these options below in the command table.

Command used:	`[dspace]/bin/dspace index-lucene-init`
Java class:	org.dspace.browse.IndexBrowse

Arguments short and long forms):	Description
`-r` or `-rebuild`	Should we rebuild all the indexes, which removes old tables and creates new ones. For use with `-f`. Mutually exclusive with `-d`
`-s` or `-start`	`-s <int>` start from this index number and work upwards (mostly only useful for debugging). For use with `-t` and `-f`
`-x` or `-execute`	Execute all the remove and create SQL against the database. For use with `-t` and `-f`
`-i` or `-index`	Actually do the indexing. Mutually exclusive with `-t` and `-f`.
`-o` or `-out`	`-o <filename>` write the remove and create SQL to the given file. For use with `-t` and `-f`
`-p` or `-print`	Write the remove and create SQL to the stdout. For use with `-t` and `-f`.
`-t` or `-tables`	Create the tables only, do no attempt to index. Mutually exclusive with `-f` and `-i`
`-f` or `-full`	Make the tables, and do the indexing. This forces `-x`. Mutually exclusive with `-f` and `-i`.
`-v` or `-verbose`	Print extra information to the stdout. If used in conjunction with `-p`, you cannot use the stdout to generate your database structure.
`-d` or `-delete`	Delete all the indexes, but do not create new ones. For use with `-f`. This is mutually exclusive with `-r`.
`-h` or `-help`	Show this help documentation. Overrides all other arguments.

If you are using the Solr Browse DAOs, that is the default since DSpace 4.0, it is not required to run this script as the data are stored in the Solr search core that need to be recreated using the Discovery maintenance script

Running the Indexing Programs

Complete Index Regeneration

Requires that you stop Tomcat first

Because this command actually **deletes** existing Browse Index tables, you **must** stop Tomcat (or your Servlet Container of choice) before executing `index-lucene-init`. After the indexing command completes, you can restart Tomcat.

Known Oracle Issues

In many **Oracle** based DSpace installations, index-lucene-init often malfunctions because of Oracle specific permissions. It is therefore advised to stick to index-lucene-update instead

By running `[dspace]/bin/dspace index-lucene-init` you will completely regenerate your indexes, tearing down all existing tables and reconstructing with the new configuration.

```
[dspace]/bin/dspace index-lucene-init
```

Updating the Indexes

By running `[dspace]/bin/dspace index-lucene-update` you will reindex your full browse & search indexes without modifying the DSpace table structure. (This should be your default approach if indexing, for example, via a cron job periodically). Because it does not "tear down" the existing tables, this command can be run while DSpace (and Tomcat or similar) is still running.

```
[dspace]/bin/dspace index-lucene-update
```

If you are using the Solr Browse DAOs, that is the default since DSpace 4.0, you don't need to run this script as the data are stored in the Solr search core. You need to recreate the indexes using the Discovery maintenance script

Destroy and Rebuild Browse Tables

This is really not recommended unless you know what you are doing.

You can destroy and rebuild the database, but do not do the indexing. Output the SQL to do this to the screen and a file, as well as executing it against the database, while being verbose.

At the CLI screen:

```
[dspace]/bin/dspace index-db-browse -r -t -p -v -x -o myfile.sql
```

Indexing Customization

Browse Index Customization

DSpace provides robust browse indexing. It is possible to expand upon the default indexes delivered at the time of the installation. The System Administrator should review Browse Index Configuration to become familiar with the property keys and the definitions used therein before attempting heavy customizations.

Through customization is is possible to:

- Add new browse indexes besides the four that are delivered upon installation. Examples:
  - Series
  - Specific subject fields (Library of Congress Subject Headings). *(It is possible to create a browse index based on a controlled vocabulary or thesaurus.)*
  - Other metadata schema fields
- Combine metadata fields into one browse
- Combine different metadata schemas in one browse

Examples of new browse indexes that are possible. *(The system administrator is reminded to read the section on Browse Index Configuration)*

- **Add a Series Browse**. You want to add a new browse using a previously unused metadata element.
  - `webui.browse.index.6 = series:metadata:dc.relation.ispartofseries:text:single`
  - Note: the index # need to be adjusted to your browse stanza in the _dspace.cfg_ file. Also, you will need to update your *Messages.properties* file.
- **Combine more than one metadata field into a browse.**You may have other title fields used in your repository. You may only want one or two of them added, not all title fields. And/or you may want your series to file in there.
  - `webui.browse.index.3 = title:metadata:dc.title,dc:title.uniform,dc:relation.ispartofseries:title:full`
- **Separate subject browse.**You may want to have a separate subject browse limited to only one type of subject.
  - `webui.browse.index.7 = lcsubject.metdata:dc.subject.lcsh.text:single`

As one can see, the choices are limited only by your metadata schema, the metadata, and your imagination.

Because Browse Indexes are stored in database tables, remember to run `index-lucene-init` after adding any new definitions in the `dspace.cfg` to have the indexes created and the data indexed.

Since DSpace 4.0 the Solr DAOs implementation of the browse engine is used by default you don't need to run the script described in this page at least if you have re-enabled the legacy DBMS provider. Instead use the Discovery maintenance script. Browse indexing in Solr is done within the Search Indexing process.

Search Index Customization

Please note, that as of DSpace 4.0, the Solr-based Discovery search is on by the default in both JSPUI and XMLUI. If you want customize the search behavior in a normal DSpace you should refer to the Discovery documentation.

Configuring Lucene Search Indexes

Search indexes can be configured and customized easily in the *dspace.cfg* file. This allows institutions to choose which DSpace metadata fields are indexed by Lucene.

Property:	`search.dir`
Example Value:	`search.dir = ${dspace.dir}/search`
Informational Note:	Where to put the search index files
Property:	`search.max-clauses`

Example Value:	`search.max-clauses = 2048`
Informational Note:	By setting higher values of search.max-clauses will enable prefix searches to work on larger repositories.
Property:	`search.index.delay`
Example Value:	`search.index.delay = 5000`
Informational Note:	It is possible to create a 'delayed index flusher'. If a web application pushes multiple search requests (i.e. a barrage or sword deposits, or multiple quick edits in the user interface), then this will combine them into a single index update. You set the property key to the number of milliseconds to wait for an update. The example value will hold a Lucene update in a queue for up to 5 seconds. After 5 seconds all waiting updates will be written to the Lucene index.
Property:	`search.analyzer`
Example Value:	`search.analyzer = org.dspace.search.DSAnalyzer`
Informational Note:	Which Lucene Analyzer implementation to use. If this is omitted or commented out, the standard DSpace analyzer (designed for English) is used by default. This standard DSpace analyzer removes common stopwords, lowercases all words and performs stemming (removing common word endings, like "ing", "s", etc).
Property:	`search.analyzer`
Example Value:	`search.analyzer = org.dspace.search.DSNonStemmingAnalyzer`
Informational Note:	Instead of the standard DSpace Analyzer (DSAnalyzer), use an analyzer which doesn't "stem" words/terms. When using this analyzer, a search for "wellness" will always return items matching "wellness" and not "well". However, similarly a search for "experiments" will only return objects matching "experiments" and not "experiment" or "experimenting". When using this analyzer, you may still use WildCard searches like "experiment*" to match the beginning of words.
Property:	`search.analyzer`
Example Value:	`search.analyzer = org.apache.lucene.analysis.cn.ChineseAnalyzer`

Informational Note:	Instead of the standard English analyzer, the Chinese analyzer is used.
Property:	`search.operator`
Example Value:	`search.operator = OR`
Informational Note	Boolean search operator to use. The currently supported values are OR and AND. If this configuration item is missing or commented out, OR is used. AND requires all the search terms to be present. OR requires one or more search terms to be present.
Property:	`search.maxfieldlength`
Example Value:	`search.maxfieldlength = 10000`
Informational Note:	This is the maximum number of terms indexed for a single field in Lucene. The default is 10,000 words, often not enough for full-text indexing. If you change this, you will need to re-index for the change to take effect on previously added items. *-1* = unlimited (Integer.MAG_VALUE)
Property:	`search.index.` *n*
Example Value:	`search.index.1 = author:dc.contributor.*`
Informational Note	This property determines which of the metadata fields are being indexed for search. As an example, if you do not include the title field here, searching for a word in the title will not be matched with the titles of your items..

For example, the following entries appear in the default DSpace installation:

```
search.index.1 = author:dc.contributor.*
search.index.2 = author:dc.creator.*
search.index.3 = title:dc.title.*
search.index.4 = keyword:dc.subject.*
search.index.5 = abstract:dc.description.abstract
search.index.6 = author:dc.description.statementofresponsibility
search.index.7 = series:dc.relation.ispartofseries
search.index.8 = abstract:dc.description.tableofcontents
search.index.9 = mime:dc.format.mimetype
search.index.10 = sponsor:dc.description.sponsorship
search.index.11 = id:dc.identifier.*
search.index.12 = language:dc.language.iso
```

The format of each entry is `search.index.<id> = <search index name> : <schema> . <metadata field>[:index type]` where:

`<id>`	is an incremental number to distinguish each search index entry
`<search index name>`	is the identifier for the search field this index will correspond to
`<schema>`	is the schema used. Dublin Core (DC) is the default. Others are possible.

`<metadata field>`	is the DSpace metadata field to be indexed.
`<index type>`	can be used to specify how manipulate the values before indexing. *Example: search.index.12 = language:dc.language.iso:inputform* Possible values are: **text** - default, no special treatment. Metadata value are passed to lucene as text **timestamp** - the values are interpreted as date with second granularity. An additional index postfixed with .year is created with year granularity **date** - the values are interpreted as date with day granularity. An additional index postfixed with .year is created with year granularity **inputform** - in addition to the values stored in the metadata the displayed form of this value as derivable from the input-form (in any of the available languages) are stored

In the example above, `search.index.1` and `search.index.2` and `search.index.3` are configured as the `author` search field. The `author` index is created by Lucene indexing all `dc.contributor.*`,`dc.creator.*` and `description.statementofresponsibility` metadata fields.

After changing the configuration run `/[dspace]/bin/dspace index-lucene-init` to regenerate the indexes.

While the indexes are created, this only affects the search results and has no effect on the search components of the user interface.

In the above examples, notice the asterisk (`*`). The metadata field (at least for Dublin Core) is made up of the "element" and the "qualifier". The asterisk is used as the "wildcard". So, for example, `keyword.dc.subject.*` will index all subjects regardless if the term resides in a qualified field. (subject versus subject.lcsh). One could customize the search and only index LCSH (Library of Congress Subject Headings) with the following entry `keyword:dc.subject.lcsh` *instead of*`keyword:dc.subject.*`

Authority Control Note:

Although DSIndexer automatically builds a separate index for the authority keys of any index that contains authority-controlled metadata fields, the "Advanced Search" UIs do not allow direct access to it. Perhaps it will be added in the future. Fortunately, the OpenSearch API lets you submit a query directly to the Lucene search engine, and this may include the authority-controlled indexes.

Customize the advanced search form

As the previous configuration apply only to the indexing and querying phase one will need to customize the user interface to reflect the changes, for example, to add the a new search category to the Advanced Search.

XML UI requires manual coding of the involved templates instead the JSP UI provides specific configuration to set the index to show in the advanced search dropdown. Below are listed the configuration parameters

Property:	`jspui.search.index.display.<n>`
Example Value	`jspui.search.index.display.1 = ANY`
Informational Note:	Set the *N*-value of the index dropdown in the advanced search form. The value must match one of the defined index