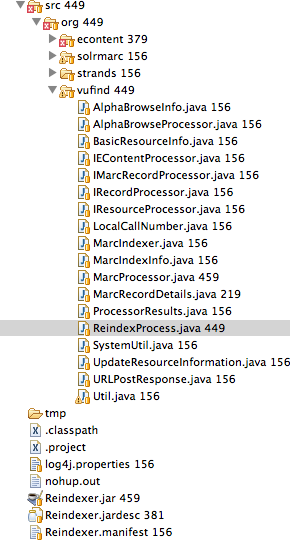
## How to run the Re-index process

cd into the vufind\import directory and then execute the following command

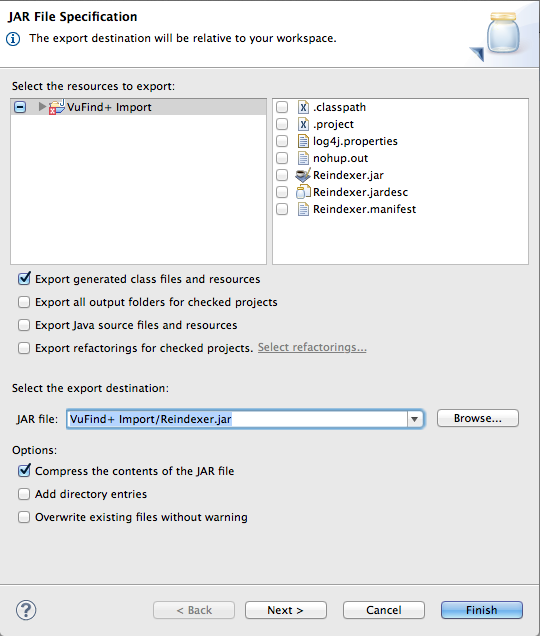
**java –jar Reindexer.jar servername**

## How do I build the Reindexer.jar or modify it?

The whole re-index process is written in Java. The source code is in the vufind/import folder.



When you make a change to the Java code, you need to rebuild the Reindexer.jar. In Eclipse, all you have to do is **double click on the Reindexer.jardesc**

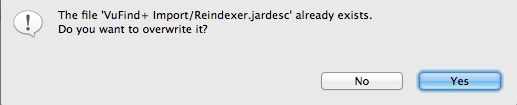


**Then click on Finish.**

You may get a message that Reindexer.jar will be overwritten. **Just click Yes**.

**IMPORTANT**: if you get a message saying that Reindexer.jardesc already exists, like this:

**It is best to click No.**



## What does the Reindexer.jar do?

When you run the Reindexer.jar, the class ReindexProccess (see ReindexProccess.java) is executed. There are 8 main steps:

**Step 1: Load configuration files**

The configuration from config.ini is loaded. Options for the [Reindex] process are grouped under the [Reindex] section of the file. If there are any options you want to override, you can put them into a separate .ini file in the conf directory and pass the file name to the Reindexer.jar as the second argument.

**NOTE**: Before any work is done, a log entry for the process is created in **reindex\_log** table in the vufind database so results can be viewed in the web administration interface.

**Step 2: Run MARC export script**

If there is a script to export MARC from ILS, you can set the **extractScript** option in config.ini file. If the **extractScript** option is set, it will be executed at this point.

**Step 3: Reload schema.xml**

If the option **reloadDefaultSchema** is set to true in config.ini, the **schema.xml** files for **biblio**, **biblio2** and **econtent** Solr cores are copied from the sites/default/solr and overwriting the site specific files. If you customize any **schema.xml**, you should set this option to false to avoid your changes from being overwritten.

**Step 4: Load record processors**

There are currently 5 “record processors” in use (ie: classes that implement **IRecordProcessor** interface) – **MarcIndexer**, **UpdateResourceInformation**, **ExtractEContentFromMarc**, **StrandsProcessor**, and **AlphaBrowseProcessor**.

In addition to implementing **IRecordProcessor** interface, the following classes also implement **IMarcRecordProcessor** interface: **MarcIndexer**, **UpdateResourceInformation, ExtractEContentFromMarc**, **StrandsProcessor**.

Depending on options in config.ini, various record processors are loaded. For DCL, all record processors are loaded.

If **updateSolr** is set, then **MarcIndexer** class (MarcIndexer.java) is loaded.

If **updateResources** is set, then **UpdateResourceInformation** class (src/org/vufind/UpdateResourcesInformation.java) is loaded.

If **loadEContentFromMarc** is set, then **ExtractEContentFromMarc** class (src/org/econtent/ExtractEContentFromMarc.java) is loaded.

If **exportStrandsCatalog** is set, then **StrandsProcessor** class (src/org/strands/StrandsProcessor.java) is loaded.

If **updateAlphaBrowse** is set, then **AlphaBrowseProcessor** class (src/org/vufind/AlphaBrowseProcessor.java) is loaded.

**IMPORTANT**: if no record processor is loaded, then the re-index process will stop here. For DCL, all processors are loaded.

**Stage 5: Process Marc files**

At this point, **MarcProcessor** class (src/org/vufind/MarcProcessor.java) is loaded, and the **init** method is executed. The following is done during **MarcProcessor** initialization:

* A new **ProcessorResults** record is created to store results of the process, the process name is “Process MARC Files”. This is the process name we see in the Administrator interface.
* Load checksums for all MARC records that had been processed before from **marc\_import** table in vufind database.
* Load eContent detection rules from **econtent\_record\_detection\_settings** table in econtent database.
* Load eContent link detection rules for all libraries from the **library** table in vufind database. The library settings can be configured in the web administrator interface.
* Load ratings for print records from **user\_rating** table in vufind database
* Load ratings for eContent records from **econtent\_rating** table in econtent database.

After the initialization is done, the **processMarcFiles** method is executed to process Marc files in the directory specified by **marcRecordPath** in config.ini. The **active\_econtent\_records** table is truncated (emptied out). All files with extension .mrc or .marc are processed one-by-one.

**IMPORTANT**: in order to automatically suppress/delete eContent records that no longer exists in any Marc files, a table active\_econtent\_records is created in the econtent Mysql database. This table is used to store IDs and ILS IDs of eContent records that exist in the Marc files. Each time an eContent record is encountered in a Marc file, its eContent record ID and ILS ID is saved into the active\_econtent\_records table. Finally, when all the Marc files have been processed, eContent records that are NOT in the active\_econtent\_records table are suppressed/deleted.

Each Marc file is processed in alphabetical order based on file name.

**IMPORTANT**: because of the nature of the process, if there are duplicate records in the Marc export files, then the records that are processed last overwrite the ones seen earlier.

Each Marc record is processed one-by-one. For each Marc record, the following is done:

* Check whether the record is NEW, CHANGED, or UNCHANGED
* Pass the Marc record to the record processors (**MarcIndexer**, **UpdateResourceInformation, ExtractEContentFromMarc**, **StrandsProcessor**) for processing one-by-one. How each of these record processors work is explained later in this document.

After all the record has been processed by all the record processors, checksums are calculated and updated/inserted into the **marc\_import** table.

When all Marc files are processed, the **suppressInActiveEContentRecords** method is called to perform suppression/deletion of eContent records that no longer exists in any of the exported Marc files. This process checks every active eContent record against the **active\_econtent\_records** table, if the record is not in the **active\_econtent\_records** table, it is added to the list of eContent records to be suppressed/deleted.

As a safety measure, the number of eContent records to be suppressed/deleted is compared against the total number of eContent records. If the percentage of records to be suppressed/deleted is less than the configuration option **eContentSuppressionThreshold,** then the suppression/deletion is performed. Otherwise, no record is suppressed/deleted.

**Step 6: Process eContent records that have been imported into econtent database**

After eContent records are extracted from Marc files and inserted/updated in **econtent\_record** table, additional processing are done by record processors that implement the **IEContentProcessor** interface. Currently on 2 record processors implement this interface: **UpdateResourceInformation** and **StrandsProcessor**.

First **processEContentRecord** in **UpdateResourceInformation** class is called to process each eContent record in the **econtent\_record** table. This process cleans up the ISBN and UPC code for each eContent record. It also creates the sort value for the title of each eContent record by removing the leading articles.

Then the **processEContentRecord** in **StrandsProcessor** is called to process each eContent record in the **econtent\_record** table to create a CVS file for export to Strands.

**Step 7: Process resources in database**

Currently the only record processor that implements the **IResourceProcessor** interface is the **AlphaBrowseProcessor** class. However, the **processResource** method does NOTHING. All processing is actually done in the **finish** method. See explanation of the **AlphaBrowseProcessor** below.

**Step 8: Finishing**

The last step in the re-index process is to execute the **finish** method in each record processor. The finish methods are called in the following on order: **MarcIndexer**, **UpdateResourceInformation**, **ExtractEContentFromMarc**, **StrandsProcessor**, and finally on **AlphaBrowseProcessor**. See the finish step for each record processor below.

Finally, save the log entries for the process into the database so they can be viewed in the administrator web interface.

**What does MarcIndexer do?**

**Initialization:**

Initialization is done exactly once at the start of the whole re-index process.

Create an entry in the re-index log called “Update Solr” which can be viewed in the administrator interface later.

Connect to the **biblio2** core on Solr server at **localhost**.

**IMPORTANT**: **localhost** is hardcoded in this processor

If **clearMarcRecordsAtStartOfIndex** is set to true in config.ini, then delete ALL records from **biblio2** core and ALL records are re-indexed.

**IMPORTANT**: if **clearMarcRecordsAtStartOfIndex** is set to true, then ALL records are re-indexed regardless of the **reindexUnchangedRecords** setting.

**Process Marc Record:**

Check if the record has changed. If it is unchanged and the **reindexUnchangedRecords** is not set to true or **clearMarcRecordsAtStartOfIndex** is not set to true, then this processor will not process it.

Check if the record is an eContent record. If the record is eContent, then this processor will not process it.

If the record is NOT eContent, then the Marc record is mapped into into a Solr XML record based on the rules in **marc\_local.properties** configuration file, and then added to the **biblio2** core.

**Finish:**

Send Solr **commit** and **optimize** messages to the **biblio2** core on **localhost** Solr server. NOTE: the **localhost** is hardcoded here.

After the **biblio2** core is optimized, check the number of errors. If less than 1% of the records have error, then the **biblio2** is swapped with **biblio** core so changes become visible to the public searches. If more than 1% of records have errors, then no swapping is done, and the new updates are not visible by the public searches.

Lastly, save the log entries of the “Update Solr” process to the database.

**What does UpdateResourceInformation do?**

**Initialization:**

Initialization is done exactly once at the start of the whole re-index process.

Create an entry in the re-index log called “Update Resources” which can be viewed in the administrator interface later.

Load resources from the **resource** table into a “hash map” in memory. This is done so that whenever a Marc record is processed, the corresponding record in the hash map is removed. What this means is that, when all Marc records have been processed, records that are still in the hash map no longer exist in the Marc export files.

**Process Marc Record:**

Check if the record has changed. If it is unchanged and the **reindexUnchangedRecords** is not set to true or **clearMarcRecordsAtStartOfIndex** is not set to true, then this processor will not process it.

Check if the record is an eContent record. If the record is eContent, then this processor will not process it.

If the record is not eContent record, then this processor extracts title, author, isbn, format etc… then insert/update the record in the **resource** table in vufind mysql database. The call numbers for the record is stored in the **resource\_callnumber** table and the subjects are stored in the **resource\_subject** table.

**Finish:**

If the **removeTitlesNotInMarcExport** option is set in config.ini, then records that are no longer in the Marc files are flagged as “deleted” in the **resource** table.

Finally all log entries for the “Update Resources” process are saved to the database.

**What does ExtractEContentFromMarc do?**

This processor extracts data from eContent Marc records, save the extracted information to the **econtent\_record** and **econtent\_item** table in eContent Mysql database, then indexes the record into the **econtent** Solr core.

**Initialization:**

Initialization is done exactly once at the start of the whole re-index process.

Create an entry in the re-index log called “Extract eContent from ILS” which can be viewed in the administrator interface later.

**Process Marc Record:**

Check if the record is an eContent record. If the record is NOT eContent, then this processor will not process it.

If the record is eContent, then the following is done:

Load the detection setting based on the source of the record. These settings can be managed in the web Administrator interface.

If there is no detection setting for the record, an error entry is logged and this record is NOT processed.

Normally a record only has 1 source, but theoretically it could have more than one. This processor allows for the possibility of the record has more than one source. For each source, the following is done:

* Get the detection settings based on the source of the record
* Check the access type of the record. If the access type is “overdrive”, and the **checkOverDriveAvailability** option is set to true in config.ini, then the record will be processed regardless of whether it is changed or unchanged. This is done to ensure that the OverDrive availability is updated.
* Check for ILS ID in the record, if no ILS ID is present, an error message is logged.
* Use the ILS ID to check if the record has been imported into the **econtent\_record** table already. If the record was imported already, then update the information in the database. If the record was never imported before, then insert the new record into the database.
* After a NEW eContent record has been added into the **econtent\_record** table. This processor inserts the ILS ID of the record into the **active\_econtent\_records** table. Then, items records are created in **econtent\_item** table based on the rules set in the detection settings. There are 3 types of eContent records: OverDrive, Guttenberg and External Link. Records whose source is not OverDrive or Guttenberg are treated as External Links. See detailed explanation below.
* Finally, the eContent record is re-index in **econtent** Solr core. See explanation below

**Finish:**

Wait a maximum of 5 minutes for all the eContent indexing threads to finish.

Finally, save log entries for the “Extract eContent from ILS” process into the database.

**How are eContent links detected?**

**MarcRecordDetails** is responsible for extracting data from Marc records.

The **loadUrls** is responsible for detecting eContent links. It looks for predefined “hints” from the 856 subfields y, z, and 3, for example “Access eBook”. It also looks for certain patterns in the 856 fields, for example “ebook.3m.com”. Once a link is detected, it will use the detection rules configured in the library system configuration to determine which library owns the link. If that information cannot be determined, it will look in the item tags (usually 949) for the library system to associate the link with that library.

**How does ExtractEContentFromMarc create OverDrive items?**

Given a list of links that were detected by the **MarcRecordDetails** class, it will check each link for OverDrive ID pattern, if the link does not match an OverDrive link, then it is ignored. If the link matches the OverDrive pattern, then the OverDrive ID is extracted from the link. The OverDrive ID is then checked in the **econtent\_item** table to see if it exists. If the OverDrive ID exists, then the new information is updated. If the OverDrive ID does not exist, then a new eContent item is added to the **econtent\_item** table.

**How does ExtractEContentFromMarc create Guttenberg items?**

Given a list of links that were detected by the **MarcRecordDetails** class, it will check each link with the list of all Guttenberg items loaded from the .CSV file and then check in the **econtent\_item** table to see if an item has been created or not. If an item had already been created, the information is updated. If the item is new, then it is added to the **econtent\_item** table.

**How does ExtractEContentFromMarc create External Link items?**

Given a list of links that were detected by the **MarcRecordDetails** class, it will check each link with **econtent\_item** table for the given eContent record. If there is any link for the eContent record that is not in the detected list of links, then they are deleted. If there are any links that are not yet in the **econtent\_item** table, they are added to the **econtent\_item** table.

**How does ExtractEContentFromMarc index econtent Solr core?**

An eContent record is indexed in the **econtent** Solr core by making a request to the URL [http://vufind-url/EcontentRecord/**eContentID**/Reindex](http://vufind-url/EcontentRecord/eContentID/Reindex), where **eContentID** is the eContent record id. You can index an eContent record manually by going to that URL using a web browser.

During the re-index process, many eContent records need to be re-indexed. Each record is indexed in a separate thread. A **EContentReindexThread** is created for each record. A maximum of 5 concurrent threads are created due to the configuration in the Solr server. This means only 5 records can be re-indexed concurrently. This is why the process takes a long time to complete.

**What does StrandsProcessor do?**

**Initialization:**

Initialization is done exactly once at the start of the whole re-index process.

Create an entry in the re-index log called “Strands Export” which can be viewed in the administrator interface later.

**Process Marc Record:**

Extract title, author, publisher, format etc… and append into the CSV file for export to strands.

**Finish:**

Save and close the CSV file for export to Strands.

**What does AlphaBrowseProcessor do?**

Run queries to create alphabetic browse tables from resources table.

**Initialization:**

Initialization is done exactly once at the start of the whole re-index process.

Create an entry in the re-index log called “Alpha Browse Table Update” which can be viewed in the administrator interface later.

**Process Resource:**

Does NOTHING. All work is actually done in the Finish method.

**Finish:**

Truncate the **title\_browse** table.

Select number of resources grouped by **title\_sort** field from the **resource** table and insert the results into the **title\_browse** table.

Truncate the **author\_browse** table.

Select number of resources grouped by **author** field from the **resource** table and insert the results into the **author\_browse** table.

Truncate the **subject\_browse** table.

Join **resource** table with **resource\_subject** table and select the number of resources grouped by subject and insert the results into the **subject\_browse** table.

Truncate the **callnumber\_browse** table.

Join **resource** table with **resource\_callnumber** and select number of resources grouped by callnumber and insert into the **callnumber\_browse** table.

Finally, save log entries for the “Alpha Browse Table Update” process into the database.