System Administration

DSpace System Documentation: System Administration

DSpace operates on several levels: as a Tomcat servlet, cron jobs, and on-demand operations. This section explains many of the on-demand operations. Some of the command operations may be also set up as cron jobs. Many of these operations are performed at the Command Line Interface (CLI) also known as the Unix prompt (`$:`). Future reference will use the term CLI when the use needs to be at the command line.

Below is the "Command Help Table". This table explains what data is contained in the individual command/help tables in the sections that follow.

Command used:	*The directory and where the command is to be found.*
Java class:	*The actual java program doing the work.*
Arguments:	*The required/mandatory or optional arguments available to the user.*

DSpace Command Launcher

With DSpace Release 1.6, the many commands and scripts have been replaced with a simple `[dspace]/bin/dspace <command>` command. See Application Layer chapter for the details of the DSpace Command Launcher.

Table of Contents:

Community and Collection Structure Importer

This CLI tool gives you the ability to import a community and collection structure directory from a source XML file.

Command used:	`[dspace]/bin/dspace structure-builder`
Java class:	`org.dspace.administer.StructBuilder`
Argument: short and long (if available) forms:	Description of the argument
-f	Source xml file.
-o	Output xml file.
-e	Email of DSpace Administrator.

The administrator need to build the source xml document in the following format:

```
<import_structure>
        <community>
                <name>Community Name</name>
                <description>Descriptive text</description>
                <intro>Introductory text</intro>
                <copyright>Special copyright notice</copyright>
                <sidebar>Sidebar text</sidebar>
                <community>
                   <name>Sub Community Name</name>
                   <community> ...[ad infinitum]...
                   </community>
                </community>
                <collection>
                   <name>Collection Name</name>
                   <description>Descriptive text</description>
                   <intro>Introductory text</intro>
                   <copyright>Special copyright notice</copyright>
                   <sidebar>Sidebar text</sidebar>
                   <license>Special licence</license>
                   <provenance>Provenance information</provenance>
                </collection>
         </community>
</import_structure>
```

The resulting output document will be as follows:

```
<import_structure>
     <community identifier="123456789/1">
                <name>Community Name</name>
                <description>Descriptive text</description>
                <intro>Introductory text</intro>
                <copyright>Special copyright notice</copyright>
                <sidebar>Sidebar text</sidebar>
                <community identifier="123456789/2">
                        <name>Sub Community Name</name>
                        <community identifier="123456789/3"> ...[ad infinitum]...
                        </community>
                </community>
                <collection identifier="123456789/4">
                        <name>Collection Name</name>
                        <description>Descriptive text</description>
                        <intro>Introductory text</intro>
                        <copyright>Special copyright notice</copyright>
                        <sidebar>Sidebar text</sidebar>
                        <license>Special licence</license>
                        <provenance>Provenance information</provenance>
                </collection>
        </community>
</import_structure>
```

This command-line tool gives you the ability to import a community and collection structure directly from a source XML file. It is executed as follows:

```
[dspace]/bin/dspace structure-builder -f /path/to/source.xml -o path/to/output.xml -e admin@user.com
```

This will examine the contents of *source.xml*, import the structure into DSpace while logged in as the supplied administrator, and then output the same structure to the output file, but including the handle for each imported community and collection as an attribute.

Limitation

- Currently this does not export community and collection structures, although it should only be a small modification to make it do so

Package Importer and Exporter

This command-line tool gives you access to the Packager plugins. It can *ingest* a package to create a new DSpace Object (Community, Collection or Item), or *disseminate* a DSpace Object as a package.

To see all the options, invoke it as:

```
[dspace]/bin/dspace packager --help
```

This mode also displays a list of the names of package ingestion and dissemination plugins that are currently installed in your DSpace. Each Packager plugin also may allow for custom options, which may provide you more control over how a package is imported or exported. You can see a listing of all specific packager options by invoking `--help` (or `-h`) with the `--type` (or `-t`) option:

```
[dspace]/bin/dspace packager --help --type METS
```

The above example will display the normal help message, while also listing any additional options available to the "METS" packager plugin.

Ingesting

Ingestion Modes & Options

When ingesting packages DSpace supports several different "modes". (Please note that not all packager plugins may support all modes of ingestion)

1. Submit/Ingest Mode (`-s` option, default) – submit package to DSpace in order to create a new object(s)
2. Restore Mode (`-r` option) – restore pre-existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "submit", where the object is created with a known Handle and known relationships.
3. Replace Mode (`-r -f` option) – replace existing object(s) in DSpace based on package(s). This also attempts to restore all handles and relationships (parent/child objects). This is a specialized type of "restore" where the contents of existing object(s) is replaced by the contents in

the AIP(s). By default, if a normal "restore" finds the object already exists, it will back out (i.e. rollback all changes) and report which object already exists.

Ingesting a Single Package

To ingest a single package from a file, give the command:

```
[dspace]/bin/dspace packager -e [user-email] -p [parent-handle] -t [packager-name] /full/path/to/package
```

Where *[user-email]* is the e-mail address of the E-Person under whose authority this runs; *[parent-handle]* is the Handle of the Parent Object into which the package is ingested, *[packager-name]* is the plugin name of the package ingester to use, and */full/path/to/package* is the path to the file to ingest (or *"-"* to read from the standard input).

Here is an example that loads a PDF file with internal metadata as a package:

```
[dspace]/bin/dspace packager -e admin@myu.edu -p 4321/10 -t PDF thesis.pdf
```

This example takes the result of retrieving a URL and ingests it:

```
wget -O - http://alum.mit.edu/jarandom/my-thesis.pdf | [dspace]/bin/dspace packager -e admin@myu.edu -p 4321
/10  -t PDF -
```

Ingesting Multiple Packages at Once

Some Packager plugins support bulk ingest functionality using the `--all` (or `-a`) flag. When `--all` is used, the packager will attempt to ingest all child packages referenced by the initial package (and continue on recursively). Some examples follow:

- For a Site-based package - this would ingest **all** Communities, Collections & Items based on the located package files
- For a Community-based package - this would ingest that Community and all SubCommunities, Collections and Items based on the located package files
- For a Collection - this would ingest that Collection and all contained Items based on the located package files
- For an Item – this just ingest the Item (including all Bitstreams & Bundles) based on the package file.

Here is a basic example of a bulk ingest 'packager' command template:

```
[dspace]/bin/dspace packager -s -a -t AIP -e <eperson> -p <parent-handle> <file-path>
```

for example:

```
[dspace]/bin/dspace packager -s -a -t AIP -e admin@myu.edu -p 4321/12 collection-aip.zip
```

The above command will ingest the package named "collection-aip.zip" as a child of the specified Parent Object (handle="4321/12"). The resulting object is assigned a new Handle (since `-s` is specified). In addition, any child packages directly referenced by "collection-aip.zip" are also recursively ingested (a new Handle is also assigned for each child AIP).

Not All Packagers Support Bulk Ingest

⚠ Because the packager plugin must know how to locate all child packages from an initial package file, not all plugins can support bulk ingest. Currently, in DSpace the following Packager Plugins support bulk ingest capabilities:

- METS Packager Plugin
- AIP Packager Plugin

Restoring/Replacing using Packages

Restoring is slightly different than just **ingesting**. When restoring, the packager makes every attempt to restore the object as it **used to be** (including its handle, parent object, etc.).

There are currently three restore modes:

1. Default Restore Mode (`-r`) = Attempt to restore object (and optionally children). Rollback all changes if any object is found to already exist.
2. Restore, Keep Existing Mode (`-r -k`) = Attempt to restore object (and optionally children). If an object is found to already exist, skip over it (and all children objects), and continue to restore all other non-existing objects.
3. Force Replace Mode (`-r -f`) = Restore an object (and optionally children) and **overwrite** any existing objects in DSpace. Therefore, if an object is found to already exist in DSpace, its contents are replaced by the contents of the package. *WARNING: This mode is potentially dangerous as it will permanently destroy any object contents that do not currently exist in the package. You may want to first perform a backup, unless you are sure you know what you are doing!*

Default Restore Mode

By default, the restore mode (`-r` option) will rollback all changes if any object is found to already exist. The user will be informed if which object already exists within their DSpace installation.

Use this 'packager' command template:

```
[dspace]/bin/dspace packager -r -t AIP -e <eperson> <file-path>
```

For example:

```
[dspace]/bin/dspace packager -r -t AIP -e admin@myu.edu aip4567.zip
```

Notice that unlike `-s` option (for submission/ingesting), the `-r` option does not require the Parent Object (`-p` option) to be specified if it can be determined from the package itself.

In the above example, the package "aip4567.zip" is restored to the DSpace installation with the Handle provided within the package itself (and added as a child of the parent object specified within the package itself). If the object is found to already exist, all changes are rolled back (i.e. nothing is restored to DSpace)

Restore, Keep Existing Mode

When the "Keep Existing" flag (`-k` option) is specified, the restore will attempt to skip over any objects found to already exist. It will report to the user that the object was found to exist (and was not modified or changed). It will then continue to restore all objects which do not already exist. This flag is most useful when attempting a bulk restore (using the `--all` (or `-a`) option.

One special case to note: If a Collection or Community is found to already exist, its child objects are also skipped over. So, this mode will not auto-restore items to an existing Collection.

Here's an example of how to use this 'packager' command:

```
[dspace]/bin/dspace packager -r -a -k -t AIP -e <eperson> <file-path>
```

For example:

```
[dspace]/bin/dspace packager -r -a -k -t AIP -e admin@myu.edu aip4567.zip
```

In the above example, the package "aip4567.zip" is restored to the DSpace installation with the Handle provided within the package itself (and added as a child of the parent object specified within the package itself). In addition, any child packages referenced by "aip4567.zip" are also recursively restored (the `-a` option specifies to also restore all child pacakges). They are also restored with the Handles & Parent Objects provided with their package. If any object is found to already exist, it is skipped over (child objects are also skipped). All non-existing objects are restored.

Force Replace Mode

When the "Force Replace" flag (`-f` option) is specified, the restore will **overwrite** any objects found to already exist in DSpace. In other words, existing content is deleted and then replaced by the contents of the package(s).

Potential for Data Loss

Because this mode actually **destroys** existing content in DSpace, it is potentially dangerous and may result in data loss! It is recommended to always perform a full backup (assetstore files & database) before attempting to replace any existing object(s) in DSpace.

Here's an example of how to use this 'packager' command:

```
[dspace]/bin/dspace packager -r -f -t AIP -e <eperson> <file-path>
```

For example:

```
[dspace]/bin/dspace packager -r -f -t AIP -e admin@myu.edu aip4567.zip
```

In the above example, the package "aip4567.zip" is restored to the DSpace installation with the Handle provided within the package itself (and added as a child of the parent object specified within the package itself). In addition, any child packages referenced by "aip4567.zip" are also recursively ingested. They are also restored with the Handles & Parent Objects provided with their package. *If any object is found to already exist, its contents are replaced by the contents of the appropriate package.*

If any error occurs, the script attempts to rollback the entire replacement process.

Disseminating

Disseminating a Single Object

To disseminate a single object as a package, give the command:

```
[dspace]/bin/dspace packager -d -e [user-email] -i [handle] -t [packager-name] [file-path]
```

Where *[user-email]* is the e-mail address of the E-Person under whose authority this runs; *[handle]* is the Handle of the Object to disseminate; *[packager-name]* is the plugin name of the package disseminator to use; and *[file-path]* is the path to the file to create (or *"-"* to write to the standard output). For example:

```
[dspace]/bin/dspace packager -d -t METS -e admin@myu.edu -i 4321/4567 4567.zip
```

The above code will export the object of the given handle (4321/4567) into a METS file named "4567.zip".

Disseminating Multiple Objects at Once

To export an object hierarchy, use the `-a` (or `--all`) package parameter.

For example, use this 'packager' command template:

```
[dspace]/bin/dspace packager -d -a -e [user-email] -i [handle] -t [packager-name][file-path]
```

for example:

```
[dspace]/bin/dspace packager -d -a -t METS -e admin@myu.edu -i 4321/4567 4567.zip
```

The above code will export the object of the given handle (4321/4567) into a METS file named "4567.zip". In addition it would export all children objects to the same directory as the "4567.zip" file.

Archival Information Packages (AIPs)

As of DSpace 1.7, DSpace now can backup and restore all of its contents as a set of AIP Files. This includes all Communities, Collections, Items, Groups and People in the system.

This feature came out of a requirement for DSpace to better integrate with DuraCloud (http://www.duracloud.org), and other backup storage systems. One of these requirements is to be able to essentially "backup" local DSpace contents into the cloud (as a type of offsite backup), and "restore" those contents at a later time.

Essentially, this means DSpace can export the entire hierarchy (i.e. bitstreams, metadata and relationships between Communities/Collections/Items) into a relatively standard format (a METS-based, AIP format). This entire hierarchy can also be re-imported into DSpace in the same format (essentially a restore of that content in the same or different DSpace installation).

For more information, see the section on AIP backup & Restore for DSpace.

METS packages

Since DSpace 1.4 release, the software includes a package disseminator and matching ingester for the DSpace METS SIP (Submission Information Package) format. They were created to help end users prepare sets of digital resources and metadata for submission to the archive using well-defined standards such as METS, MODS, and PREMIS. The plugin name is *METS* by default, and it uses MODS for descriptive metadata.

The DSpace METS SIP profile is available at: https://wiki.duraspace.org/display/DSPACE/DSpaceMETSSIPProfile

Item Importer and Exporter

DSpace has a set of command line tools for importing and exporting items in batches, using the DSpace simple archive format. The tools are not terribly robust, but are useful and are easily modified. They also give a good demonstration of how to implement your own item importer if desired.

DSpace Simple Archive Format

The basic concept behind the DSpace's simple archive format is to create an archive, which is directory full of items, with a subdirectory per item. Each item directory contains a file for the item's descriptive metadata, and the files that make up the item.

```
archive_directory/
    item_000/
        dublin_core.xml         -- qualified Dublin Core metadata for metadata fields belonging to the dc schema
        metadata_[prefix].xml   -- metadata in another schema, the prefix is the name of the schema as
registered with the metadata registry
        contents                -- text file containing one line per filename
        file_1.doc              -- files to be added as bitstreams to the item
        file_2.pdf
    item_001/
        dublin_core.xml
        contents
        file_1.png
        ...
```

The *dublin_core.xml* or *metadata*[prefix].xml_file has the following format, where each metadata element has it's own entry within a *<dcvalue>* tagset. There are currently three tag attributes available in the *<dcvalue>* tagset:

- *<element>* - the Dublin Core element
- *<qualifier>* - the element's qualifier
- *<language>* - (optional)ISO language code for element

```
<dublin_core>
    <dcvalue element="title" qualifier="none">A Tale of Two Cities</dcvalue>
    <dcvalue element="date" qualifier="issued">1990</dcvalue>
    <dcvalue element="title" qualifier="alternate" language="fr">J'aime les Printemps</dcvalue>
</dublin_core>
```

(Note the optional language tag attribute which notifies the system that the optional title is in French.)

Every metadata field used, must be registered via the metadata registry of the DSpace instance first.

The *contents* file simply enumerates, one file per line, the bitstream file names. See the following example:

```
        file_1.doc
        file_2.pdf
        license
```

Please notice that the *license* is optional, and if you wish to have one included, you can place the file in the .../item_001/ directory, for example.

The bitstream name may optionally be followed by any of the following:

- *\tbundle:BUNDLENAME*
- *\tpermissions:PERMISSIONS*
- *\tdescription:DESCRIPTION*
- *\tprimary:true*

Where '\t' is the tab character.

'BUNDLENAME' is the name of the bundle to which the bitstream should be added. Without specifying the bundle, items will go into the default bundle, ORIGINAL.

'PERMISSIONS' is text with the following format: -[r|w] 'group name'

'DESCRIPTION' is text of the files description.

Primary is used to specify the primary bitstream.

Configuring *metadata-[prefix].xml* for Different Schema

It is possible to use other Schema such as EAD, VRA Core, etc. Make sure you have defined the new scheme in the DSpace Metada Schema Registry.

1. Create a separate file for the other schema named "*metadata-[prefix].xml*", where the *{prefix}* is replaced with the schema's prefix.
2. Inside the xml file use the dame Dublin Core *syntax*, but on the *<dublin_core>* element include the attribute "*schema={prefix}*".
3. Here is an example for ETD metadata, which would be in the file "*metadata_etd.xml*":

```
<?xml version="1.0" encoding="UTF-8"?>
<dublin_core schema="etd">
     <dcvalue element="degree" qualifier="department">Computer Science</dcvalue>
     <dcvalue element="degree" qualifier="level">Masters</dcvalue>
     <dcvalue element="degree" qualifier="grantor">Texas A & M</dcvalue>
</dublin_core>
```

Importing Items

Before running the item importer over items previously exported from a DSpace instance, please first refer to Transferring Items Between DSpace Instances.

Command used:	`[dspace]/bin/dspace import`
Java class:	`org.dspace.app.itemimport.ItemImport`
Arguments short and (long) forms:	Description
`-a or --add`	Add items to DSpace ‡
`-r or --replace`	Replace items listed in mapfile ‡
`-d or --delete`	Delete items listed in mapfile ‡
`-s or --source`	Source of the items (directory)
`-c or --collection`	Destination Collection by their Handle or database ID
`-m or --mapfile`	Where the mapfile for items can be found (name and directory)
`-e or --eperson`	Email of eperson doing the importing
`-w or --workflow`	Send submission through collection's workflow
`-n or --notify`	Kicks off the email alerting of the item(s) has(have) been imported
`-t or --test`	Test run, do not actually import items
`-p or --template`	Apply the collection template
`-R or --resume`	Resume a failed import (Used on Add only)
`-h or --help`	Command help

‡ These are mutually exclusive.

The item importer is able to batch import unlimited numbers of items for a particular collection using a very simple CLI command and 'arguments'

Adding Items to a Collection

To add items to a collection, you gather the following information:

- eperson
- Collection ID (either Handle (e.g. 123456789/14) or Database ID (e.g. 2)
- Source directory where the items reside
- Mapfile. Since you don't have one, you need to determine where it will be (e.g. /Import/Col_14/mapfile)
  At the command line:

```
[dspace]/bin/dspace import --add --eperson=joe@user.com --collection=CollectionID --source=items_dir --
mapfile=mapfile
```

or by using the short form:

```
[dspace]/bin/dspace import -a -e joe@user.com -c CollectionID -s items_dir -m mapfile
```

The above command would cycle through the archive directory's items, import them, and then generate a map file which stores the mapping of item directories to item handles. **SAVE THIS MAP FILE.** Using the map file you can use it for replacing or deleting (unimporting) the file.

Testing. You can add `--test` (or `-t`) to the command to simulate the entire import process without actually doing the import. This is extremely useful for verifying your import files before doing the actual import.

Replacing Items in Collection

Replacing existing items is relatively easy. Remember that mapfile you were *supposed* to save? Now you will use it. The command (in short form):

```
[dspace]/bin/dspace import -r -e joe@user.com -c collectionID -s items_dir -m mapfile
```

Long form:

```
[dspace]/bin/dspace import --replace --eperson=joe@user.com --collection=collectionID --source=items_dire --
mapfile=mapfile
```

Deleting or Unimporting Items in a Collection

You are able to unimport or delete items provided you have the mapfile. Remember that mapfile you were *supposed* to save? The command is (in short form):

```
[dspace]/bin/dspace import -d -m mapfile
```

In long form:

```
[dspace]/bin/dspace import --delete --mapfile mapfile
```

Other Options

- **Workflow**. The importer usually bypasses any workflow assigned to a collection. But add the `--workflow` (`-w`) argument will route the imported items through the workflow system.
- **Templates**. If you have templates that have constant data and you wish to apply that data during batch importing, add the `--template` (`-p`) argument.
- **Resume**. If, during importing, you have an error and the import is aborted, you can use the `--resume` (`-R`) flag that you can try to resume the import where you left off after you fix the error.

Exporting Items

The item exporter can export a single item or a collection of items, and creates a DSpace simple archive for each item to be exported.

Command used:	`[dspace]/bin/dspace export`
Java class:	*org.dspace.app.itemexport.ItemExport*
Arguments short and (long) forms:	Description
`-t` or `--type`	Type of export. *COLLECTION* will inform the program you want the whole collection. *ITEM* will be only the specific item. (You will actually key in the keywords in all caps. See examples below.)
`-i` or `--id`	The ID or Handle of the Collection or Item to export.
`-d` or `--dest`	The destination of where you want the file of items to be placed. You place the path if necessary.
`-n` or `--number`	Sequence number to begin export the items with. Whatever number you give, this will be the name of the first directory created for your export. The layout of the export is the same as you would set your layout for an Import.
`-m` or `--migrate`	Export the item/collection for migration. This will remove the handle and metadata that will be re-created in the new instance of DSpace.
`-h` or `--help`	Brief Help.

Exporting a Collection

To export a collection's items you type at the CLI:

```
[dspace]/bin/dspace export --type=COLLECTION --id=collID --dest=dest_dir --number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t COLLECTION -d CollID or Handle -d /path/to/destination -n Some_number
```

Exporting a Single Item

The keyword *COLLECTION* means that you intend to export an entire collection. The ID can either be the database ID or the handle. The exporter will begin numbering the simple archives with the sequence number that you supply. To export a single item use the keyword *ITEM* and give the item ID as an argument:

```
[dspace]/bin/dspace export --type=ITEM --id=itemID --dest=dest_dir --number=seq_num
```

Short form:

```
[dspace]/bin/dspace export -t ITEM -i itemID or Handle -d /path/to/destination -n some_number
```

Each exported item will have an additional file in its directory, named 'handle'. This will contain the handle that was assigned to the item, and this file will be read by the importer so that items exported and then imported to another machine will retain the item's original handle.

The `-m` Argument

Using the `-m` argument will export the item/collection and also perform the migration step. It will perform the same process that the next section Transferring Items Between DSpace Instances performs. We recommend that the next section be read in conjunction with this flag being used.

Transferring Items Between DSpace Instances

Migration of Data
Where items are to be moved between DSpace instances (for example from a test DSpace into a production DSpace) the item exporter and item importer can be used in conjunction with a script to assist in this process.

After running the item exporter each *dublin_core.xml* file will contain metadata that was automatically added by DSpace. These fields are as follows:

- date.accessioned
- date.available
- date.issued
- description.provenance
- format.extent
- format.mimetype
- identifier.uri
  In order to avoid duplication of this metadata, run

```
[dspace]/bin/dspace_migrate </path/to/exported item directory>
```

prior to running the item importer. This will remove the above metadata items, except for date.issued - if the item has been published or publicly distributed before and *identifier.uri* - if it is not the handle, from the *dublin_core.xml* file and remove all *handle* files. It will then be safe to run the item exporter.

Item Update

ItemUpdate is a batch-mode command-line tool for altering the metadata and bitstream content of existing items in a DSpace instance. It is a companion tool to ItemImport and uses the DSpace simple archive format to specify changes in metadata and bitstream contents. Those familiar with generating the source trees for ItemImporter will find a similar environment in the use of this batch processing tool.

For metadata, ItemUpdate can perform 'add' and 'delete' actions on specified metadata elements. For bitstreams, 'add' and 'delete' are similarly available. All these actions can be combined in a single batch run.

ItemUpdate supports an undo feature for all actions except bitstream deletion. There is also a test mode, as with ItemImport. However, unlike ItemImport, there is no resume feature for incomplete processing. There is more extensive logging with a summary statement at the end with counts of successful and unsuccessful items processed.

One probable scenario for using this tool is where there is an external primary data source for which the DSpace instance is a secondary or down-stream system. Metadata and/or bitstream content changes in the primary system can be exported to the simple archive format to be used by ItemUpdate to synchronize the changes.

A note on terminology: **item** refers to a DSpace item. **metadata element** refers generally to a qualified or unqualified element in a schema in the form *[schema].[element].[qualifier]* or *[schema].[element]* and occasionally in a more specific way to the second part of that form. **metadata field** refers to a specific instance pairing a metadata element to a value.

DSpace simple Archive Format

As with ItemImporter, the idea behind the DSpace's simple archive format is to create an archive directory with a subdirectory per item. There are a few additional features added to this format specifically for ItemUpdate. Note that in the simple archive format, the item directories are merely local references and only used by ItemUpdate in the log output.

The user is referred to the previous section DSpace Simple Archive Format.

Additionally, the use of a **delete_contents** is now available. This file lists the bitstreams to be deleted, one bitstream ID per line. Currently, no other identifiers for bitstreams are usable for this function. This file is an addition to the Archive format specifically for ItemUpdate.

The optional suppress_undo file is a flag to indicate that the 'undo archive' should not be written to disk. This file is usually written by the application in an undo archive to prevent a recursive undo. This file is an addition to the Archive format specifically for ItemUpdate.

ItemUpdate Commands

Command used:	`[dspace]/bin/dspace itemupdate`
Java class:	*org.dspace.app.itemimport.ItemUpdate*
Arguments short and (long) forms:	Description
`-a` or `--addmetadata [metadata element]`	Repeatable for multiple elements. The metadata element should be in the form dc.x or dc.x.y. The mandatory argument indicates the metadata fields in the dublin_core.xml file to be added unless already present. However, duplicate fields will not be added to the item metadata without warning or error.
`-d` or `--deletemetadata [metadata element]`	Repeatable for multiple elements. All metadata fields matching the element will be deleted.
`-A` or `--addbitstream`	Adds bitstreams listed in the contents file with the bitstream metadata cited there.
`-D` or `--deletebitstream [filter plug classname or alias]`	Not repeatable. With no argument, this operation deletes bitstreams listed in the *deletes_contents* file. Only bitstream ids are recognized identifiers for this operatiotn. The optional filter argument is the classname of an implementation of *org.dspace.app.itemdupate.BitstreamFilter* class to identify files for deletion or one of the aliases (ORIGINAL, ORIGINAL_AND_DERIVATIVES, TEXT, THUMBNAIL) which reference existing filters based on membership in a bundle of that name. IN this case, the *delete_contents* file is not required for any item. The filter properties file will contains properties pertinent to the particular filer used. Multiple filters are not allowed.
`-h` or `--help`	Displays brief command line help.
`-e` or `--eperson`	Email address of the person or the user's database ID **(Required)**
`-s` or `--source`	Directory archive to process **(Required)**
`-i` or `--itemidentifier`	Specifies an alternate metadata field (not a handle) used to hold an identifier used to match the DSpace item with that in the archive. If omitted, the item handle is expected to be located in the *dc.identifier.uri* field. (Optional)
`-t` or `--test`	Runs the process in test mode with logging but no changes applied to the DSpace instance. (Optional)
`-P` or `--alterprovenance`	Prevents any changes to the provenance field to represent changes in the bitstream content resulting from an Add or Delete. No provenance statements are written for thumbnails or text derivative bitstreams, in keeping with the practice of MediaFilterManager. (Optional)
`-F` or `--filterproperties`	The filter properties files to be used by the delete bitstreams action (Optional)

CLI Examples

Adding Metadata:

```
[dspace]/bin/dspace itemupdate -e joe@user.com -s [path/to/archive] -a dc.description
```

This will add from your archive the dc element description based on the handle from the URI (since the `-i` argument wasn't used).

Registering (Not Importing) Bitstreams

Registration is an alternate means of incorporating items, their metadata, and their bitstreams into DSpace by taking advantage of the bitstreams already being in storage accessible to DSpace. An example might be that there is a repository for existing digital assets. Rather than using the normal interactive ingest process or the batch import to furnish DSpace the metadata and to upload bitstreams, registration provides DSpace the metadata and the location of the bitstreams. DSpace uses a variation of the import tool to accomplish registration.

Accessible Storage

To register an item its bitstreams must reside on storage accessible to DSpace and therefore referenced by an asset store number in *dspace.cfg*. The configuration file *dspace.cfg* establishes one or more asset stores through the use of an integer asset store number. This number relates to a directory in the DSpace host's file system or a set of SRB account parameters. This asset store number is described in The *dspace.cfg* Configuration Properties File section and in the *dspace.cfg* file itself. The asset store number(s) used for registered items should generally not be the value of the *assetstore.incoming* property since it is unlikely that you will want to mix the bitstreams of normally ingested and imported items and registered items.

Registering Items Using the Item Importer

DSpace uses the same import tool that is used for batch import except that several variations are employed to support registration. The discussion that follows assumes familiarity with the import tool.

The archive format for registration does not include the actual content files (bitstreams) being registered. The format is however a directory full of items to be registered, with a subdirectory per item. Each item directory contains a file for the item's descriptive metadata (*dublin_core.xml*) and a file listing the item's content files (*contents*), but not the actual content files themselves.

The *dublin_core.xml* file for item registration is exactly the same as for regular item import.

The *contents* file, like that for regular item import, lists the item's content files, one content file per line, but each line has the one of the following formats:

```
-r -s n -f filepath
-r -s n -f filepath\tbundle:bundlename
-r -s n -f filepath\tbundle:bundlename\tpermissions: -[r|w] 'group name'
-r -s n -f filepath\tbundle:bundlename\tpermissions: -[r|w] 'group        name'\tdescription: some text
```

where

- *-r* indicates this is a file to be registered
- *-s n* indicates the asset store number (*n*)
- *-f filepath* indicates the path and name of the content file to be registered (filepath)
- \t is a tab character
- *bundle:bundlename* is an optional bundle name
- *permissions: -[r|w] 'group name'* is an optional read or write permission that can be attached to the bitstream
- *description: some text* is an optional description field to add to the file
  The bundle, that is everything after the filepath, is optional and is normally not used.

The command line for registration is just like the one for regular import:

```
[dspace]/bin/dspace import -a -e joe@user.com -c collectionID -s items_dir -m mapfile
```

(or by using the long form)

```
[dspace]/bin/dspace import --add --eperson=joe@user.com --collection=collectionID --source=items_dir --
map=mapfile
```

The `--workflow` and `--test` flags will function as described in Importing Items.

The `--delete` flag will function as described in Importing Items but the registered content files will not be removed from storage. See Deleting Registered Items.

The `--replace` flag will function as described in Importing Items but care should be taken to consider different cases and implications. With old items and new items being registered or ingested normally, there are four combinations or cases to consider. Foremost, an old registered item deleted from DSpace using `--replace` will not be removed from the storage. See Deleting Registered Items. where is resides. A new item added to DSpace using `--replace` will be ingested normally or will be registered depending on whether or not it is marked in the *contents* files with the `-r`.

Internal Identification and Retrieval of Registered Items

Once an item has been registered, superficially it is indistinguishable from items ingested interactively or by batch import. But internally there are some differences:

First, the randomly generated internal ID is not used because DSpace does not control the file path and name of the bitstream. Instead, the file path and name are that specified in the *contents* file.

Second, the *store_number* column of the bitstream database row contains the asset store number specified in the *contents* file.

Third, the *internal_id* column of the bitstream database row contains a leading flag (*-R*) followed by the registered file path and name. For example, *-Rfilepath* where *filepath* is the file path and name relative to the asset store corresponding to the asset store number. The asset store could be traditional storage in the DSpace server's file system or an SRB account.

Fourth, an MD5 checksum is calculated by reading the registered file if it is in local storage. If the registerd file is in remote storage (say, SRB) a checksum is calculated on just the file name! This is an efficiency choice since registering a large number of large files that are in SRB would consume substantial network resources and time. A future option could be to have an SRB proxy process calculate MD5s and store them in SRB's metadata catalog (MCAT) for rapid retrieval. SRB offers such an option but it's not yet in production release.

Registered items and their bitstreams can be retrieved transparently just like normally ingested items.

Exporting Registered Items

Registered items may be exported as described in Exporting Items. If so, the export directory will contain actual copies of the files being exported but the lines in the contents file will flag the files as registered. This means that if DSpace items are "round tripped" (see Transferring Items Between DSpace Instances) using the exporter and importer, the registered files in the export directory will again registered in DSpace instead of being uploaded and ingested normally.

METS Export of Registered Items

The METS Export Tool can also be used but note the cautions described in that section and note that MD5 values for items in remote storage are actually MD5 values on just the file name.

Deleting Registered Items

If a registered item is deleted from DSpace, either interactively or by using the *~~-delete~~* or *-replace* flags described in Importing Items, the item will disappear from DSpace but it's registered content files will remain in place just as they were prior to registration. Bitstreams not registered but added by DSpace as part of registration, such as *license.txt* files, will be deleted.

METS Tools

These Tools are Obsolete

These METS Export Tools are now Obsolete. They have been replaced by the METS Packager Plugin tools, which are able to both export and import METS files. See the Package Importer and Exporter section for more details.

The experimental (incomplete) METS export tool writes DSpace items to a filesystem with the metadata held in a more standard format based on METS.

The Export Tool

This tool is obsolete. Its use is strongly discouraged. Please use the Package Importer and Exporter instead.

The following are examples of the types of process the METS tool can provide.

Exporting an individual item. From the CLI:

```
[dspace]/bin/dspace org.dspace.app.mets.METSExport -i [handle] -d /path/to/destination
```

Exporting a collection. From the CLI:

```
[dspace]/bin/dspace org.dspace.app.mets.METSExport -c [handle] -d /path/to/destination
```

Exporting all the items in DSpace. From the CLI:

```
[dspace]/bin/dspace org.dspace.app.mets.METSExport -a -d /path/to/destination
```

Limitations

- No corresponding import tool yet
- No *structmap* section
- Some technical metadata not written, e.g. the primary bitstream in a bundle, original filenames or descriptions.
- Only the MIME type is stored, not the (finer grained) bitstream format.
- Dublin Core to MODS mapping is very simple, probably needs verification

MediaFilters: Transforming DSpace Content

DSpace can apply filters to content/bitstreams, creating new content. Filters are included that extract text for **full-text searching**, and create **thumbnails** for items that contain images. The media filters are controlled by the *MediaFilterManager* which traverses the asset store, invoking the *MediaFilter* or *Forma tFilter* classes on bitstreams. The media filter plugin configuration *filter.plugins* in *dspace.cfg* contains a list of all enabled media/format filter plugins (see Configuring Media Filters for more information). The media filter system is intended to be run from the command line (or regularly as a cron task):

```
[dspace]/bin/dspace filter-media
```

With no options, this traverses the asset store, applying media filters to bitstreams, and skipping bitstreams that have already been filtered.

Available Command-Line Options:

- **Help** : *[dspace]/bin/dspace filter-media -h*
  - Display help message describing all command-line options.
- **Force mode** : *[dspace]/bin/dspace filter-media -f*
  - Apply filters to ALL bitstreams, even if they've already been filtered. If they've already been filtered, the previously filtered content is overwritten.
- **Identifier mode** : *[dspace]/bin/dspace filter-media -i 123456789/2*
  - Restrict processing to the community, collection, or item named by the identifier - by default, all bitstreams of all items in the repository are processed. The identifier must be a Handle, not a DB key. This option may be combined with any other option.
- **Maximum mode** : *[dspace]/bin/dspace filter-media -m 1000*
  - Suspend operation after the specified maximum number of items have been processed - by default, no limit exists. This option may be combined with any other option.
- **No-Index mode** : *[dspace]/bin/dspace filter-media -n*
  - Suppress index creation - by default, a new search index is created for full-text searching. This option suppresses index creation if you intend to run *index-update* elsewhere.
- **Plugin mode** : *[dspace]/bin/dspace filter-media -p "PDF Text Extractor","Word Text Extractor"*
  - Apply ONLY the filter plugin(s) listed (separated by commas). By default all named filters listed in the *filter.plugins* field of *dspace.cfg* are applied. This option may be combined with any other option. *WARNING:* multiple plugin names must be separated by a comma (i.e. ',') and NOT a comma followed by a space (i.e. ', ').
- **Skip mode** : *[dspace]/bin/dspace filter-media -s 123456789/9,123456789/100*
  - SKIP the listed identifiers (separated by commas) during processing. The identifiers must be Handles (not DB Keys). They may refer to items, collections or communities which should be skipped. This option may be combined with any other option. *WARNING:* multiple identifiers must be separated by a comma (i.e. *',*') and NOT a comma followed by a space (i.e. *', '*).
  - NOTE: If you have a large number of identifiers to skip, you may maintain this comma-separated list within a separate file (e.g. *filter-skiplist.txt*). Use the following format to call the program. *Please note the use of the "grave" or "tick" (`_) symbol and do not use the single quotation. _*
    - *[dspace]/bin/dspace filter-media -s `less filter-skiplist.txt`*
- **Verbose mode** : *[dspace]/bin/dspace filter-media -v*
  - Verbose mode - print all extracted text and other filter details to STDOUT.
    Adding your own filters is done by creating a class which *implements* the *org.dspace.app.mediafilter.FormatFilter* interface. See the Creating a new Media Filter topic and comments in the source file FormatFilter.java for more information. In theory filters could be implemented in any programming language (C, Perl, etc.) However, they need to be invoked by the Java code in the Media Filter class that you create.

Sub-Community Management

DSpace provides an administrative tool, 'CommunityFiliator', for managing community sub-structure. Normally this structure seldom changes, but prior to the 1.2 release sub-communities were not supported, so this tool could be used to place existing pre-1.2 communities into a hierarchy. It has two operations, either establishing a community to sub-community relationship, or dis-establishing an existing relationship.

The familiar parent/child metaphor can be used to explain how it works. Every community in DSpace can be either a 'parent' community, meaning it has at least one sub-community, or a 'child' community, meaning it is a sub-community of another community, or both or neither. In these terms, an 'orphan' is a community that lacks a parent (although it can be a parent); 'orphans' are referred to as 'top-level' communities in the DSpace user-interface, since there is no parent community 'above' them. The first operation, establishing a parent/child relationship - can take place between any community and an orphan. The second operation - removing a parent/child relationship, will make the child an orphan.

Command used:	`[dspace]/bin/dspace community-filiator`
Java class:	*org.dspace.administer.CommunityFiliator*
Arguments short and (long) forms:	Description
`-s` or `--set`	Set a parent/child relationship
`-r` or `--remove`	Remove a parent/child relationship
`-c` or `--child`	Child community (Handle or database ID)
`-p` or `--parent`	Parent community (Handle or database ID
`-h` or `--help`	Online help.

Set a parent/child relationship, issue the following at the CLI:

```
dspace community-filiator --set --parent=parentID --child=childID
```

(or using the short form)

```
[dspace]/bin/dspace community-filiator -s -p parentID -c childID
```

where '~~s' or '~~set' means establish a relationship whereby the community identified by the '-p' parameter becomes the parent of the community identified by the '-c' parameter. Both the 'parentID' and 'childID' values may be handles or database IDs.

The reverse operation looks like this:

```
[dspace]/bin/dspace community-filiator --remove --parent=parentID --child=childID
```

(or using the short form)

```
[dspace]/bin/dspace community-filiator -r -p parentID -c childID
```

where '~~r' or '~~-remove' means dis-establish the current relationship in which the community identified by 'parentID' is the parent of the community identified by 'childID'. The outcome will be that the 'childID' community will become an orphan, i.e. a top-level community.

If the required constraints of operation are violated, an error message will appear explaining the problem, and no change will be made. An example in a removal operation, where the stated child community does not have the stated parent community as its parent: "Error, child community not a child of parent community".

It is possible to effect arbitrary changes to the community hierarchy by chaining the basic operations together. For example, to move a child community from one parent to another, simply perform a 'remove' from its current parent (which will leave it an orphan), followed by a 'set' to its new parent.

It is important to understand that when any operation is performed, all the sub-structure of the child community follows it. Thus, if a child has itself children (sub-communities), or collections, they will all move with it to its new 'location' in the community tree.

Batch Metadata Editing

DSpace provides a batch metadata editing tool. The batch editing tool is able to produce a comma delimited file in the CVS format. The batch editing tool facilitates the user to perform the following:

- Batch editing of metadata (e.g. perform an external spell check)
- Batch additions of metadata (e.g. add an abstract to a set of items, add controlled vocabulary such as LCSH)
- Batch find and replace of metadata values (e.g. correct misspelled surname across several records)
- Mass move items between collections
- Enable the batch addition of new items (without bitstreams) via a CSV file
- Re-order the values in a list (e.g. authors)

Export Function

The following table summarizes the basics.

Command used:	`[dspace]/bin/dspace metadata-export`
Java class:	org.dspace.app.bulkedit.MetadataExport
Arguments short and (long) forms):	Description
`-f` or `--file`	Required. The filename of the resulting CSV.
`-i` or `--id`	The Item, Collection, or Community handle or Database ID to export. If not specified, **all** items will be exported.
`-a` or `--all`	Include all the metadata fields that are not normally changed (e.g. provenance) or those fields you configured in the `dspace.cfg` to be ignored on export.
`-h` or `--help`	Display the help page.

Exporting Process

To run the batch editing exporter, at the command line:

```
[dspace]/bin/dspace metadata-export -f name_of_file.csv -i 1023/24
```

Example:

```
[dspace]/bin/dspace metadata-export -f /batch_export/col_14.csv -i /1989.1/24
```

In the above example we have requested that a collection, assigned handle '*1989.1/24*' export the entire collection to the file '*col_14.cvs*' found in the *'/batch_export*' directory.

Import Function

The following table summarizes the basics.

Command used:	`[dspace]/bin/dspace metadata-import`
Java class:	org.dspace.app.bulkedit.MetadataImport
Arguments short and (long) forms:	Description
`-f` or `--file`	Required. The filename of the CSV file to load.
`-s` or `--silent`	Silent mode. The import function does not prompt you to make sure you wish to make the changes.
`-e` or `--email`	The email address of the user. This is only required when adding new items.
`-w` or `--workflow`	When adding new items, the program will queue the items up to use the Collection Workflow processes.
`-n` or `--notify`	when adding new items using a workflow, send notification emails.
`-t` or `--template`	When adding new items, use the Collection template, if it exists.
`-h` or `--help`	Display the brief help page.

Silent Mode should be used carefully. It is possible (and probable) that you can overlay the wrong data and cause irreparable damage to the database.

Importing Process

To run the batch importer, at the command line:

```
[dspace]/bin/dspace metadata-import -f name_of_file.csv
```

Example

```
[dspace]/bin/dspace metadata-import -f /dImport/col_14.csv
```

If you are wishing to upload new metadata **without** bitstreams, at the command line:

```
[dspace]/bin/dspace/metadata-import -f /dImport/new_file.csv -e joe@user.com -w -n -t
```

In the above example we threw in all the arguments. This would add the metadata and engage the workflow, notification, and templates to all be applied to the items that are being added.

Importing large CSV files

It is not recommended to import CSV files of more than 1,000 lines. When importing files larger than this, it is hard to accurately verify the changes that the import tool states it will make, and large files may cause 'Out Of Memory' errors part way through the process.

The CSV Files

The csv files that this tool can import and export abide by the RFC4180 CSV format http://www.ietf.org/rfc/rfc4180.txt. This means that new lines, and embedded commas can be included by wrapping elements in double quotes. Double quotes can be included by using two double quotes. The code does all this for you, and any good csv editor such as Excel or OpenOffice will comply with this convention.

File Structure. The first row of the csv must define the metadata values that the rest of the csv represents. The first column must always be "id" which refers to the item's id. All other columns are optional. The other columns contain the dublin core metadata fields that the data is to reside.

A typical heading row looks like:

```
id,collection,dc.title,dc.contributor,dc.date.issued,etc,etc,etc.
```

Subsequent rows in the csv file relate to items. A typical row might look like:

```
350,2292,Item title,"Smith, John",2008
```

If you want to store multiple values for a given metadata element, they can be separated with the double-pipe '||' (or another character that you defined in your _dspace.cfg _file. For example:

```
Horses||Dogs||Cats
```

Elements are stored in the database in the order that they appear in the csv file. You can use this to order elements where order may matter, such as authors, or controlled vocabulary such as Library of Congress Subject Headings.

When importing a csv file, the importer will *overlay* the data onto what is already in the repository to determine the differences. It only acts on the contents of the csv file, rather than on the complete item metadata. This means that the CSV file that is exported can be manipulated quite substantially before being re-imported. Rows (items) or Columns (metadata elements) can be removed and will be ignored. For example, if you only want to edit item abstracts, you can remove all of the other columns and just leave the abstract column. (You do need to leave the ID column intact. This is mandatory).

Editing collection membership. Items can be moved between collections by editing the collection handles in the 'collection' column. Multiple collections can be included. The first collection is the 'owning collection'. The owning collection is the primary collection that the item appears in. Subsequent collections (separated by the field separator) are treated as mapped collections. These are the same as using the map item functionality in the DSpace user interface. To move items between collections, or to edit which other collections they are mapped to, change the data in the collection column.

Adding items. New metadata-only items can be added to DSpace using the batch metadata importer. To do this, enter a plus sign '+' in the first 'id' column. The importer will then treat this as a new item. If you are using the command line importer, you will need to use the -e flag to specify the user email address or id of the user that is registered as submitting the items.

Deleting Data. It is possible to perform deletes across the board of certain metadata fields from an exported file. For example, let's say you have used keywords (dc.subject) that need to be removed *en masse*. You would leave the column (dc.subject) intact, but remove the data in the corresponding rows.

Migrating Data or Exchanging data. It is possible that you have data in one Dublin Core (DC) element and you wish to really have it in another. An example would be that your staff have input Library of Congress Subject Headings in the Subject field (dc.subject) instead of the LCSH field (dc.subject. lcsh). Follow these steps and your data is migrated upon import:

1. Insert a new column. The first row should be the new metadata element. (We will refer to it as the TARGET)
2. Select the column/rows of the data you wish to change. (We will refer to it as the SOURCE)
3. Cut and paste this data into the new column (TARGET) you created in Step 1.
4. Leave the column (SOURCE) you just cut and pasted from empty. Do not delete it.

Checksum Checker

Checksum Checker is program that can run to verify the checksum of every item within DSpace. Checksum Checker was designed with the idea that most System Administrators will run it from the cron. Depending on the size of the repository choose the options wisely.

Command used:	`[dspace]/bin/dspace checker`
Java class:	org.dspace.app.checker.ChecksumChecker
Arguments short and (long) forms):	Description
`-L` or `--continuous`	Loop continuously through the bitstreams
`-a` or `--handle`	Specify a handle to check
`-b <bitstream-ids>`	Space separated list of bitstream IDs
`-c` or `--count`	Check count
`-d` or `--duration`	Checking duration
`-h` or `--help`	Calls online help
`-l` or `--looping`	Loop once through bitstreams
`-p <prune>`	Prune old results (optionally using specified properties file for configuration
`-v` or `--verbose`	Report all processing

There are three aspects of the Checksum Checker's operation that can be configured:

- the execution mode
- the logging output
- the policy for removing old checksum results from the database
  The user should refer to Chapter 5. Configuration for specific configuration beys in the *dspace.cfg* file.

Checker Execution Mode

Execution mode can be configured using command line options. Information on the options are found in the previous table above. The different modes are described below.

Unless a particular bitstream or handle is specified, the Checksum Checker will always check bitstreams in order of the least recently checked bitstream. (Note that this means that the most recently ingested bitstreams will be the last ones checked by the Checksum Checker.)

Available command line options

- **Limited-count mode:** `[dspace]/bin/dspace checker -c` To check a specific number of bitstreams. The *-c* option if followed by an integer, the number of bitstreams to check. Example: `[dspace/bin/dspace checker -c 10` This is particularly useful for checking that the checker is executing properly. The Checksum Checker's default execution mode is to check a single bitstream, as if the option was *-c 1*
- **Duration mode:** `[dspace]/bin/dspace checker -d` To run the Check for a specific period of time with a time argument. You may use any of the time arguments below: Example: `[dspace/bin/dspace checker -d 2h` (Checker will run for 2 hours)

s	Seconds
m	Minutes
h	Hours
d	Days
w	Weeks
y	Years

  The checker will keep starting new bitstream checks for the specific durations, so actual execution duration will be slightly longer than the specified duration. Bear this in mind when scheduling checks.
- **Specific Bitstream mode:** `[dspace]/bin/dspace checker -b` Checker will only look at the internal bitstream IDs. Example: `[dspace]/bin/dspace checker -b 112 113 4567` Checker will only check bitstream IDs 112, 113 and 4567.
- **Specific Handle mode:** `[dspace]/bin/dspace checker -a` Checker will only check bitstreams within the Community, Community or the item itself. Example: `[dspace]/bin/dspace checker -a 123456/999` Checker will only check this handle. If it is a Collection or Community, it will run through the entire Collection or Community.
- **Looping mode:** `[dspace]/bin/dspace checker -l` or `[dspace]/bin/dspace checker -L` There are two modes. The lowercase 'el' (-l) specifies to check every bitstream in the repository once. This is recommended for smaller repositories who are able to loop through all their content in just a few hours maximum. An uppercase 'L' (-L) specifies to continuously loops through the repository. This is not recommended for most repository systems. **Cron Jobs**. For large repositories that cannot be completely checked in a couple of hours, we recommend the -d option in cron.
- **Pruning mode:** `[dspace]/bin/dspace checker -p` The Checksum Checker will store the result of every check in the checksum_history table. By default, successful checksum matches that are eight weeks old or older will be deleted when the -p option is used. (Unsuccessful ones will be retained indefinitely). Without this option, the retention settings are ignored and the database table may grow rather large!

Checker Results Pruning

As stated above in "Pruning mode", the checksum_history table can get rather large, and that running the checker with the -p assists in the size of the checksum_history being kept manageable. The amount of time for which results are retained in the checksum_history table can be modified by one of two methods:

1. Editing the retention policies in `[dspace]/config/dspace.cfg` See Chapter 5 Configuration for the property keys. OR
2. Pass in a properties file containing retention policies when using the -p option.To do this, create a file with the following two property keys:

   ```
   checker.retention.default = 10y
   checker.retention.CHECKSUM_MATCH = 8w
   ```

   You can use the table above for your time units. At the command line: `[dspace]/bin/dspace checker -p retention_file_name <ENTER>`

Checker Reporting

Checksum Checker uses log4j to report its results. By default it will report to a log called `[dspace]/log/checker.log`, and it will report only on bitstreams for which the newly calculated checksum does not match the stored checksum. To report on all bitstreams checked regardless of outcome, use the *-v* (verbose) command line option:

`[dspace]/bin/dspace checker -l -v` (This will loop through the repository once and report in detail about every bitstream checked.

To change the location of the log, or to modify the prefix used on each line of output, edit the `[dspace]/config/templates/log4j.properties` file and run `[dspace]/bin/install_configs`.

Cron or Automatic Execution of Checksum Checker

You should schedule the Checksum Checker to run automatically, based on how frequently you backup your DSpace instance (and how long you keep those backups). The size of your repository is also a factor. For very large repositories, you may need to schedule it to run for an hour (e.g. `-d 1h` option) each evening to ensure it makes it through your entire repository within a week or so. Smaller repositories can likely get by with just running it weekly.

Unix, Linux, or MAC OS. You can schedule it by adding a cron entry similar to the following to the crontab for the user who installed DSpace:

```
0 4 * * 0 [dspace]/bin/dspace checker -d2h -p
```

The above cron entry would schedule the checker to run the checker every Sunday at 400 (4:00 a.m.) for 2 hours. It also specifies to 'prune' the database based on the retention settings in *dspace.cfg*.

Windows OS. You will be unable to use the checker shell script. Instead, you should use Windows Schedule Tasks to schedule the following command to run at the appropriate times:

```
[dspace]/bin/dspace checker -d2h -p
```

(This command should appear on a single line).

Automated Checksum Checkers' Results

Optionally, you may choose to receive automated emails listing the Checksum Checkers' results. Schedule it to run **after** the Checksum Checker has completed its processing (otherwise the email may not contain all the results).

Command used:	`[dspace]/bin/dspace checker-emailer`
Java class:	org.dspace.checker.DailyReportEmailer
Arguments short and (long) forms):	Description
`-a` or `--All`	Send all the results (everything specified below)
`-d` or `--Deleted`	Send E-mail report for all bitstreams set as deleted for today.
`-m` or `--Missing`	Send E-mail report for all bitstreams not found in assetstore for today.
`-c` or `--Changed`	Send E-mail report for all bitstreams where checksum has been changed for today.
`-u` or `--Unchanged`	Send the Unchecked bitstream report.
`-n` or `--Not Processed`	Send E-mail report for all bitstreams set to longer be processed for today.
`-h` or `--help`	Help

You can also combine options (e.g. `-m -c`) for combined reports.

Cron. Follow the same steps above as you would running checker in cron. Change the time but match the regularity. Remember to schedule this **after** Checksum Checker has run.

Embargo

If you have implemented the Embargo feature, you will need to run it periodically to check for Items with expired embargoes and lift them.

Command used:	`[dspace]/bin/dspace embargo-lifter`
Java class:	org.dspace.embargo.EmbargoManager
Arguments short and (long) forms):	Description
`-c` or `--check`	ONLY check the state of embargoed Items, do NOT lift any embargoes
`-i` or `--identifier`	Process ONLY this handle identifier(s), which must be an Item. Can be repeated.
`-l` or `--lift`	Only lift embargoes, do NOT check the state of any embargoed items.
`-n` or `--dryrun`	Do no change anything in the data model, print message instead.
`-v` or `--verbose`	Print a line describing the action taken for each embargoed item found.
`-q` or `--quiet`	No output except upon error.
`-h` or `--help`	Display brief help screen.

You must run the Embargo Lifter task periodically to check for items with expired embargoes and lift them from being embargoed. For example, to check the status, at the CLI:

```
[dspace]/bin/dspace embargo-lifter -c
```

To lift the actual embargoes on those items that meet the time criteria, at the CLI:

```
[dspace]/bin/dspace embargo-lifter -l
```

Browse Index Creation

To create all the various browse indexes that you define in the Configuration Section (Chapter 5) 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-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.

Running the Indexing Programs

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

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

Updating the Indexes. By running `[dspace]/bin/dspace index-update` you will reindex your full browse without modifying the table structure. (This should be your default approach if indexing, for example, via a cron job periodically).

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

Destroy and rebuild. 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 \-r \-t \-p \-v \-x \-o myfile.sql
```

Indexing 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 "Defining the Indexes" from the Chapter 5. 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 Defining the Indexes in Chapter 5. 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.

Remember to run *index-init* after adding any new definitions in the *dspace.cfg* to have the indexes created and the data indexed.

DSpace Log Converter

With the release of DSpace 1.6, new statistics software component was added. DSpace's use of SOLR for statics makes it possible to have a database of statistics. This in mind, there is the issue of the older log files and how a site can use them. The following command process is able to convert the existing log files and then import them for SOLR use. The user will need to perform this only once.

The Log Converter program converts log files from dspace.log into an intermediate format that can be inserted into SOLR.

Command used:	`[dspace]/bin/dspace stats-log-converter`
Java class:	org.dspace.statistics.util.ClassicDSpaceLogConverter
Arguments short and long forms):	Description
`-i` or `-in`	Input file
`-o` or `-out`	Output file
`-m` or `-multiple`	Adds a wildcard at the end of input and output, so it would mean dspace.log* would be converted. (For example, the following files would be included because of this argument: *dspace.log, dspace.log.1, dspace.log.2, dspace.log.3,* etc.)
`-n` or `-newformat`	If the log files have been created with DSpace 1.6
`-v` or `-verbose`	Display verbose output (helpful for debugging)
`-h` or `-help`	Help

The command loads the intermediate log files that have been created by the aforementioned script into SOLR.

Command used:	`[dspace]/bin/dspace stats-log-importer`
Java class:	org.dspace.statistics.util.StatisticsImporter
Arguments (short and long forms):	Description
`-i` or `--`	input file
`-m` or `--`	Adds a wildcard at the end of the input, so it would mean dspace.log* would be imported
`-s` or `--`	To skip the reverse DNS lookups that work out where a user is from. (The DNS lookup finds the information about the host from its IP address, such as geographical location, etc. This can be slow, and wouldn't work on a server not connected to the internet.)
`-v` or `--`	Display verbose ouput (helpful for debugging)
`-l` or `--`	For developers: allows you to import a log file from another system, so because the handles won't exist, it looks up random items in your local system to add hits to instead.
`-h` or `--`	Help

Although the DSpace Log Convertor applies basic spider filtering (googlebot, yahoo slurp, msnbot), it is far from complete. Please refer to Statistics Client (8.15) for spider removal operations, after converting your old logs.

Client Statistics

Command used:	`[dspace]/bin/dspace stats-util`
Java class:	org.dspace.statistics.util.StatisticsClient
Arguments (short and long forms):	Description
`-u` or `-update-spider-files`	Update Spider IP Files from internet into `/dspace/config/spiders`. Downloads Spider files identified in *dspace.cfg* under property
	Delete Spiders in Solr By isBot Flag. Will prune out all records that have `isBot:true`

`-f` or `-delete-spiders-by-flag`	
`-i` or `-delete-spiders-by-ip`	Delete Spiders in Solr By IP Address. Will prune out all records that have IP's that match spider IPs.
`-m` or `-mark-spiders`	Update isBog Flag in Solr. Marks any records currently stored in statistics that have IP addresses matched in spiders files
`-o` or `-optimize`	Run maintenance on the SOLR index. Recommended to run daily, to prevent your applet container from running out of memory
`-h` or `-help`	Calls up this brief help table at CLI.

Notes:

The usage of these options is open for the user to choose, If they want to keep spider entires in their repository, they can just mark them using "`-m`" and they will be excluded from statistics queries when "`solr.statistics.query.filter.isBot = true`" in the `dspace.cfg`.

If they want to keep the spiders out of the solr repository, they can run just use the "`-i`" option and they will be removed immediately.

There are guards in place to control what can be defined as an IP range for a bot, in `[dspace]/config/spiders`, spider IP address ranges have to be at least 3 subnet sections in length 123.123.123 and IP Ranges can only be on the smallest subnet [123.123.123.0 - 123.123.123.255]. If not, loading that row will cause exceptions in the dspace logs and exclude that IP entry.

Test Database

This command can be used at any time to test for Database connectivity. It will assist in troubleshooting PostgreSQL and Oracle connection issues with the database.

Command used:	`[dspace]/bin/dspace test-database`
Java class:	org.dspace.storage.rdbms.DatabaseManager
Arguments (short and long forms):	Description
`-` or `--`	There are no arguments used at this time.

Moving items

It is possible for administrators to move items one at a time using either the JSPUI or the XMLUI. When editing an item, on the 'Edit item' screen select the 'Move Item' option. To move the item, select the new collection for the item to appear in. When the item is moved, it will take its authorizations (who can READ / WRITE it) with it.

If you wish for the item to take on the default authorizations of the destination collection, tick the 'Inherit default policies of destination collection' checkbox. This is useful if you are moving an item from a private collection to a public collection, or from a public collection to a private collection.

- Note: When selecting the 'Inherit default policies of destination collection' option, ensure that this will not override system-managed authorizations such as those imposed by the embargo system.

Items may also be moved in bulk by using the CSV batch metadata editor (see above).